Distributed Operating Systems Lecture


2014

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 if / when and only if / 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 if / when and only if / when a trusted program needs it?

- How you can trust your system.
- How you can assure that your system is trustworthy.
Assurance

• trust developer / company
  • reputation
  • “I know the company so I can sue them if things go wrong”

• quality assuring processes
  • e.g., independent test and development team, documentation, ...

• certification
  • trust them because some experts said they are trustworthy
  • experts ensure that the company did their testing, ...

• Examples:
  • ISO 9000
  • Common Criteria Security Evaluation
  • Arinc / DO 178b

• (formal verification)
  • mathematical proof of correctness
  • required as part of Common Criteria for EAL 7 (in parts), old BSI GISA
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)
  (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. [...]”
Outline

• Introduction

• Example Proof

• Security Policies

• Policy Enforcement Mechanisms

• Undecidability of Leakage

• Take-Grant Protection Model
Formal Verification

- abstract model
- C++ Code
- more detailed model
- Refinement
- theorems
- verification system

QED
Formal Verification

11 PY to verify a 10KLOC microkernel (seL4)
Proving Security – an Example

Operations: read, write, create / delete file, create / delete user, chmod
"Only the owner of a file or root shall obtain write permissions to a file."
Proving Security – an Example

1st ingredient: abstract system model
- captures the details that are relevant for the theorem
- abstracts away all other details
- often characterized as states + state transitions
1st ingredient: abstract system model

- states:
  \[ \Sigma := \{ (U_{\text{life}}, F_{\text{life}}, \text{owner}, \text{rights}) \} \]

\[ \sigma \in \Sigma := \{(\text{root, hermann, marcus}), \{\text{foo, bar}\}, \{(\text{bar, hermann}), (\text{foo, marcus})\}, \{(\text{hermann, bar, \{w\}}), (\text{root, foo, \{r,w\}}), (\text{marcus, foo, \{r\}})\}\]
**1st ingredient: abstract system model**

- state transitions:
  \[ C := \Sigma \rightarrow \Sigma \]

read: \( \sigma \rightarrow \sigma \)

\[
\text{delete(bar)} : \sigma \rightarrow \{ \text{root, hermann, marcus} \}, \{ \text{foo, bar} \}, \{ (\text{bar, hermann}), (\text{foo, marcus}) \}, \{ (\text{hermann, bar, w}), (\text{root, foo, r,w}), (\text{marcus, foo, r}) \}\]
1st ingredient: abstract system model

- state transitions:
  \[ C := \Sigma \rightarrow \Sigma \]

read: \( \sigma \rightarrow \sigma \)

\[ u.\text{delete}(\text{bar}) : \sigma \rightarrow \text{if } u = \text{root} \lor u = \sigma \cdot \text{owner(\text{bar}) then} \]
\[ (\{\text{root, hermann, marcus}\}, \{\text{foo, bar}\}, (\{\text{bar, hermann}\}, (\text{foo, marcus}\}), \{\text{hermann, bar, \{w\}}, (\text{root, foo, \{r,w\}}, (\text{marcus, foo, \{r\}}))) \]
\[ \text{else } \sigma \text{ endif} \]
2nd ingredient: theorem

„Only the owner of a file or root shall obtain write permissions to a file.“

vs.

„Information in a file shall origin only from the owner of a file or from root.“
2nd ingredient: theorem

„Only the owner of a file or root shall obtain write permissions to a file.“

\[ P : \Sigma \rightarrow \{ \text{true}, \text{false} \} \]

secure wrt. \( P \) if \( \sigma_0 \in P \) and \( \Sigma_{\text{reachable}} \subseteq P \)
2nd ingredient: theorem

„Only the owner of a file or root shall obtain write permissions to a file.“

\[ P : \Sigma \rightarrow \{true, false\} \]

\[ P(\sigma) := \forall f \in F_{life}, u \in U_{life}, w \in \sigma.\text{rights}(u,f) \Rightarrow \\
\quad u = \text{root} \lor u = \sigma.\text{owner}(f) \]
Proving Security – an Example

3rd ingredient: proof

Theorem: $\Sigma_{\text{reachable}} \subseteq P$

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

Operations: read, write, create / delete file, create / delete user, chmod
3rd ingredient: proof

Theorem: $\Sigma_{\text{reachable}} \subseteq P$

\[ P(\sigma) := \forall f \in F_{\text{life}}, u \in U_{\text{life}}, w \in \sigma.\text{rights}(u,f) \Rightarrow u = \text{root} \lor u = \sigma.\text{owner}(f) \]

Proof:
by induction over all traces

\[
\sigma_0 \xrightarrow{\text{u.c}} \sigma \xrightarrow{\sigma'} \sigma'' \xrightarrow{\sigma'''} \sigma''' \ldots
\]

Operations: read, write, create / delete file, create / delete user, chmod
Proving Security – an Example

3rd ingredient: proof

Theorem: $\Sigma_{\text{reachable}} \subseteq P$

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

Proof:
by induction over all traces

\[\sigma_0 \xrightarrow{u.c} \sigma \xrightarrow{u'.c'} \sigma' \xrightarrow{u''.c''} \sigma'' \xrightarrow{u'''.c'''} \sigma''' \cdots\]

Operations: read, write, create / delete file, create / delete user, chmod

induction step succeeds for read, …, delete user

but

chmod(Marcus, bar, {w})
4\textsuperscript{th} ingredient: refinement

\begin{align*}
\text{chmod}(u, f, R)(s) & := \\
\quad \text{if } u = \text{root } \vee \text{owner}(f, u) \text{ then} \\
\quad & \quad s \text{ with rights } (u, f) := R \\
\quad \text{else} \\
\quad & \quad s \\
\quad \text{endif}
\end{align*}

\text{sys_chmod:}
\begin{verbatim}
parse_parameters();
  owner = file.owner;
  if (current_thread->user == root ||
      current_thread->user == owner)
    { file->set_acl(user, rights);
  }
\end{verbatim}
Outline

• Introduction
• Example Proof
• Security Policies
• Policy Enforcement Mechanisms
• Undecidability of Leakage
• Take-Grant Protection Model
Security Policies - Definition

[Bishop: Computer Security Art and Science]

• **Security Policy**
  
  A *security policy* $P$ is a statement that partitions the states $S$ of a system into a set of authorized (or secure) states (e.g., $\Sigma_{\text{sec}} := \{ \sigma \in \Sigma \mid P(\sigma) \}$) 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 unauthorized state (i.e., $\Sigma_{\text{reachable}} \subseteq \Sigma_{\text{sec}}$).
Confidentiality

prevent unauthorized disclosure of sensitive information (prevent 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
Integrity

correctness of information or data

**Definition 1:**

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

Definition:

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: \( \leq \)

**domain:** Entity \( \rightarrow L \)

- each subject has a *security clearance*: \( \text{dom}(s) \in L \)
- each object has a *security classification*: \( \text{dom}(o) \in L \)
Bell-LaPadula Policy '73 (simple version)

Policy: \((L, \leq, \text{dom})\)

rules for reading / writing

simple security condition

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

\[ s \text{ can read } o \iff \text{dom}(o) \leq \text{dom}(s) \]

* - property

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

\[ s \text{ can write } o \iff \text{dom}(s) \leq \text{dom}(o) \]
Bell-LaPadula: Multi-Level Security Policy

**Policy:** \((L, \leq, \text{dom})\)

\(\leq\) is a partial order, \((L, \leq)\) form a lattice

---

Bundesverfassungsschutzgesetz §17 - §26:

in general, no information exchange between the BND and the Police
**Concern:** Integrity (prevent damage)

(L, ≤, dom) dual to MLS

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

- s can read o  \(\iff\) dom(s) ≤ dom(o)

- s can write o  \(\iff\) 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)
**Concern:** Integrity (prevent damage)

\((L, \leq, \text{dom})\) dual to MLS

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

- s can read o  \( \iff \text{dom}(s) \leq \text{dom}(o) \)
- if s reads o then \( \text{dom}'(s) = \min(\text{dom}(s), \text{dom}(o)) \)

- s can write o  \( \iff \text{dom}(o) \leq \text{dom}(s) \)

**Problem:** label creep

Subject clearances decrease over time
No means to “clean” a tainted subject
Confidentiality and integrity are dual and can be represented in the same lattice:

Confidentiality: \[ l_{\text{conf}} \leq h_{\text{conf}} \]
Integrity: \[ h_{\text{int}} \leq l_{\text{int}} \]
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)
* 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
- Example Proof
- Security Policies
  - Policy Enforcement Mechanisms
  - Undecidability of Leakage
  - Take-Grant Protection Model
Access Control Matrix

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

Matrix: $S \times E \times R$

Operations:
- read / write entity
- create subject / object
- destroy subject / object
- enter / delete $R$ into cell $(s,o)$
Access Control List

<table>
<thead>
<tr>
<th>Subjects</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objects</td>
<td>O</td>
</tr>
<tr>
<td>Entities</td>
<td>E = S ∪ O</td>
</tr>
<tr>
<td>Rights</td>
<td>R</td>
</tr>
</tbody>
</table>

List of S x R tuples stored with every Entity

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₁</td>
<td>r, w</td>
</tr>
<tr>
<td>S₂</td>
<td>r, w</td>
</tr>
<tr>
<td>O₁</td>
<td>r</td>
</tr>
<tr>
<td>O₂</td>
<td>-</td>
</tr>
<tr>
<td>S₁</td>
<td>r, w</td>
</tr>
<tr>
<td>S₂</td>
<td>w</td>
</tr>
<tr>
<td>S₁</td>
<td>r</td>
</tr>
<tr>
<td>S₂</td>
<td>r, w</td>
</tr>
</tbody>
</table>

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 of \( E \times R \) tuples stored with every subject

More in a few minutes
Principle of Attenuation

German: Abschwächen / Verminderung

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

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:

\[
s'.grant R into (s,o) := \begin{align*}
\text{if } & R \subseteq (s',o) \text{ then enter } R \text{ into } (s,o) 
\end{align*}
\]
Capabilities

**Definition:** unforgeable token $E \times R$

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

Definition: unforgeable token $E \times R$

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

Definition: unforgeable token $E \times R$

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

Definition: unforgeable token $E \times R$

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
Definition: unforgeable token $E \times R$

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

Definition: 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

Definition: unforgeable token $E \times R$

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

**Definition:** unforgeable token $E \times R$

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
  - **diminishing take**
  - diminishing grant

![Diagram of Capability Propagation](image-url)
Definition: unforgeable token $E \times R$

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
  - diminishing take
  - diminishing grant

Diagram:
- $S_{Alice}$
- $S_{Bob}$
- $O$

$(r, w, t, g, dt, dg)$

$\{r, dt\}$
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
Reference Monitors

More general security policies

Security policies

Nothing bad happens

System remains operational
Outline

• Introduction

• Example Proof

• Security Policies

• Policy Enforcement Mechanisms

• Undecidability of Leakage

• Take-Grant Protection Model
Leakage (of Access Rights)

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 access right $r$ 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 \( M: A, B, C, \ldots \)
state automaton \( K: x, y, z, \ldots \)
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)

$TM \cong \text{universal TM} \cong \text{while}$

**Theorem:** The halting problem is undecidable
Theorem: the halting problem is undecidable

Proof: by contradiction
assume such a program P exists; write two programs:

\[
\text{does\_P\_terminate\_on\_input\_E} (P, E) :=
\begin{align*}
&\text{if } P(E) \text{ terminates } \{ \text{return true} \} \text{ else } \{ \text{return false} \}
\end{align*}
\]

\[
\text{test} (P) := \text{while} (\text{does\_P\_terminate\_on\_input\_E}(P, P))
\]

now, if \( \text{does\_P\_terminate\_on\_input\_E}(\text{test}, \text{test}) \) returns true, \( \text{test}(\text{test}) \) must terminate \([if \ condition]\)

but then the condition of the while loop is true, which means \( \text{test}(\text{test}) \) 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 \((s,o)\) if and only if TM halts

   \[ \implies \text{leakage in the ACM could be used to solve the halting problem, which is known to be undecidable} \]

   \[ \implies \text{leakage is undecidable} \]
Simulating a TM with an ACM

δ: (x, A) -> (y, B, L)

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

\[ \delta: (x, A) \rightarrow (y, B, L) \]

Distributed Operating Systems
Marcus Völp, Hermann Härtig

June, 30th, 2014
Simulating a TM with an ACM

\[ \delta: (x, A) \rightarrow (y, B, L) \]

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

\[ \text{if } x \in (s_{head}, s_{head}) \land A \in (s_{head}, s_{head}) \quad \text{then} \]

\[ \ldots \]
Simulating a TM with an ACM

\[ \delta: (x, A) \rightarrow (y, B, L) \]

\[ c_{x,A} (s_{\text{head}}, s_{\text{left}}) := \]

\hspace{1cm}\text{if } x \in (s_{\text{head}}, s_{\text{head}}) \land A \in (s_{\text{head}}, s_{\text{head}}) \]

\hspace{1cm}\text{then delete } x, A \text{ from } (s_{\text{head}}, s_{\text{head}}) \]

\hspace{1cm} \ldots
Simulating a TM with an ACM

\( \delta: (x, A) \rightarrow (y, B, L) \)

\[ \begin{align*}
\text{c}_{x,A}(s_{\text{head}}, s_{\text{left}}) & := \\
\text{if } x \in (s_{\text{head}}, s_{\text{head}}) & \wedge \\
A \in (s_{\text{head}}, s_{\text{head}}) & \text{ then} \\
\text{delete } x,A \text{ from } (s_{\text{head}}, s_{\text{head}}) & \\
\text{enter } B \text{ into } (s_{\text{head}}, s_{\text{head}}) & \\
\text{enter } y \text{ into } (s_{\text{left}}, s_{\text{left}}) & \\
\ldots & 
\end{align*} \]
Simulating a TM with an ACM

\[ \delta: (x, A) \rightarrow (y, B, L) \]

\[ c_{x,A}(s_{\text{head}}, s_{\text{left}}) := \ldots \]

x is leaked into (si,si) \[ \Leftrightarrow \]

TM halts in x
Simulating a TM with an ACM

Problem 1:

How to detect if we are at the last cell?
Simulating a TM with an ACM

Problem 1:

How to detect if we are at the last cell?

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td></td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td></td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td></td>
<td></td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>S4</td>
<td></td>
<td></td>
<td></td>
<td>D,x,e</td>
</tr>
</tbody>
</table>
Problem 2:

How do we restrict the ACM to only execute the TM program?

\[ c_{x,A}(s, s') := \ldots \]

applies to all \( s, s' \) pairs; not only neighboring
Problem 2:

How do we restrict the ACM to only execute the TM program?

c_{x,A} (s, s') :=

... applies to all s, s' pairs; not only neighboring
Outline

• Introduction
• Example Proof
• Security Policies
• Policy Enforcement Mechanisms
• Undecidability of Leakage
• Take-Grant Protection Model
Take Grant Protection Model

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

Transition Rules:

- Take

- Grant

- Create

- Remove

- Diminish
A few Lemmas:

- Take

- Lemma 1:

- Grant

- Lemma 2:
A few Lemmas:

- Lemma 3:

\[
\begin{align*}
&x \quad \quad y \quad \quad z \\
&t \quad \quad g \quad \quad t^*
\end{align*}
\]
Proof of Lemma 1

\[
\begin{align*}
\text{x.create} & \quad v \ (tg) \\
\text{y.take} & \quad g \text{ on } v \\
\text{y.grant} & \quad \beta \text{ on } z \text{ to } v \\
\text{x.take} & \quad \beta \text{ on } z \text{ from } v
\end{align*}
\]

Lemmas 2 and 3 are left for the exercises
Theorem:

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

Proof Sketch:

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:

\[ \vdash^* \text{ by Lemma 1} \]
Example:

\[
\begin{align*}
\text{Example:} & \quad \text{by Lemma 1} \\
\text{Example:} & \quad \text{by Lemma 3}
\end{align*}
\]
Example:

\[ \vdash^* \text{by Lemma 1} \]

\[ \vdash^* \text{by Lemma 3} \]

\[ x.\text{grant } \beta \text{ on } z \text{ to } w \]

\[ u.\text{take } \beta \text{ on } z \text{ from } w \]
Islands and bridges: leakage in TG is decidable in linear time

- need to consider only t,g edges for building the graph
- Lemmas 1, 2 \(\Rightarrow\) t v g edge between subjects \(\Rightarrow\) full rights exchange
Islands and bridges: towards deciding leakage in linear time

- need to consider only $t,g$ edges for building the graph
- Lemmas 1, 2 $\Rightarrow t \lor g$ edge between subjects $\Rightarrow$ full rights exchange
Islands and bridges: towards deciding leakage in linear time

- need to consider only t,g edges for building the graph
- Lemmas 1, 2 => t v g edge between subjects => full rights exchange

Bridge: $t^n g t^m$
Summary

- Certification
  - Assuring system security

- Verification Example

- Security Policies
  - Confidentiality (MLS), Integrity (Biba), mixed (Chinese Wall)

- Policy Enforcement Mechanisms
  - ACLs, Capabilities, Monitors

- Undecidability of Leakage
  - ACM implements turing machine

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

- B. Lampson: A note on the confinement problem
- **Matt Bishop – Text Book:** Computer Security – Art and Science
- 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 enforceable 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