Non-interference

Christoph Sprenger and Sebastian Mödersheim
Department of Computer Science
ETH Zurich

FMSEC Module 11, v.2
November 30, 2009

Outline

Limitations of Access Control Models

- AC models (such as AC matrix model, RBAC, Bell-LaPadula, Biba) restrict operations like read and write.
- But information may be revealed in other ways.
- Examples of information leaks:
  - ★ Error messages to user, e.g., “file not found”.
  - ★ Timing behavior based on CPU usage.
    To send a 0-bit use 100% of CPU, to send a 1-bit, sleep.
  - ★ Locking and unlocking files.
  - ★ Encode information in the invoice sent for services rendered.
- Such covert channels first noticed by Butler Lampson in 1973.
- Cryptographic techniques can be used to make these channels almost impossible to detect (Gustavus Simmons, 1993).

Interface Models as Alternative

- Rather than restricting operations, interface models specify restrictions on systems' input/output. This is more abstract.
- Leaves open problem of defining what is visible at the interface. E.g., timing? Power consumption?
- But which properties do we demand of the interface? Example: interface should not reveal my credit card number.
- The idea of non-interference is really rather simple: a security domain u is non-interfering with domain v if no input by u can influence subsequent outputs seen by v.
- Non-interference specifies a security property: The absence of illicit information flow with respect to a policy specifying the allowed information flows between domains.
- Up to implementer to design an enforcement mechanism.

Outline

Systems

We model a computer system by a conventional finite-state automaton.

Definition (System)

A system (or machine) is composed of

- a set S of *states*, with an initial state $s_0 \in S$,
- a set A of *actions*, and
- a set O of *outputs*,

together with the functions *step* and *output*:

- $step : S \times A \to S$,
- $output : S \times A \to O$.

- Actions can be thought of as "inputs," or "commands," or "instructions" to be performed by the machine.
- $step(s, a)$ denotes *the* next state of the system when action a is applied in state s. The systems we consider here are deterministic.
- $output(s, a)$ denotes the result returned by the action.

From Steps to Runs

We extend the function step to sequences of actions.

Definition (Functions *run*, *do* and *test*)

We define the function $run : S \times A^* \to S$ by

$$\begin{aligned} run(s, \epsilon) &= s \\ run(s, a \cdot \alpha) &= run(step(s, a), \alpha) \end{aligned}$$

We also define the auxiliary functions

- $do(\alpha) = run(s_0, \alpha)$ and
- $test(\alpha, a) = output(do(\alpha), a)$.

- $run(s, \alpha)$ returns the state resulting from the application of the actions sequence α to s.
- Since the system is deterministic, this result is uniquely determined by the sequence α.
- $do(\alpha)$ applies α to the initial state s_0.

Holly, Lucy and the 2-Bit Machine

Example (Definition of 2-bit machine)

Sets

- set of states $S = \mathit{Bool} \times \mathit{Bool}$,
- set of actions $A = U \times C$,
- set of users $U = \{\text{Holly}, \text{Lucy}\}$,
- set of commands $C = \{\text{flip}, \text{skip}\}$,
- set of outputs $O = \mathit{Bool} \times \mathit{Bool} \cup \mathit{Bool}$.

Steps and Output

$$\mathit{step}((h,l),(u,\text{flip})) = (\overline{h},\overline{l}) \qquad \mathit{output}((h,l),(\text{Holly},c)) = (h,l)$$
$$\mathit{step}((h,l),(u,\text{skip})) = (h,l) \qquad \mathit{output}((h,l),(\text{Lucy},c)) = l$$

Run $\alpha = (\text{Holly},\text{skip})(\text{Lucy},\text{flip})(\text{Holly},\text{flip})$

- $\mathit{do}(\alpha) = \mathit{run}(s_0,\alpha) = (0,1)$ for $s_0 = (0,1)$
- $\mathit{test}(\alpha,(\mathit{Lucy},\mathit{skip})) = \mathit{output}(\mathit{do}(\alpha),(\mathit{Lucy},\mathit{skip})) = 1$

Security Policies

- In order to discuss security, we must assume a set of security "domains" and a policy that restricts the allowable flow of information among those domains.
- The agents or subjects of a particular security domain interact with the system by presenting it with actions, and observing the results.
- Thus we assume a set D of security domains, and
- a function $dom : A \rightarrow D$ associating a security domain with each action.

Definition (Security policy)

A security policy is specified by a reflexive relation $\rightsquigarrow$ on D.

- We use $\not\rightsquigarrow$ to denote the complement relation, i.e.,$(D \times D) \setminus \rightsquigarrow$.
- We speak of $\rightsquigarrow$ and $\not\rightsquigarrow$ as the interference and noninterference relations.
- A policy is said to be transitive if its interference relation has that property.

Purging Actions

- We wish to define security in terms of information flow, so the next step is to capture the idea of the "flow of information" formally.
- The key observation is that information can be said to flow from a domain u to a domain v exactly when actions submitted by domain u cause the behavior of the system perceived by domain v to be different from that perceived when those actions are not present.

Definition (Function *purge*)

For $v \in D$ and $\alpha \in A^*$, we define $purge(\alpha, v)$ to be the subsequence of α formed by deleting all actions associated with domains u such that $u \not\rightsquigarrow v$:

$$
\begin{aligned}
purge(\epsilon, v) &= \epsilon \\
purge(a \cdot \alpha, v) &= \begin{cases} a \cdot purge(\alpha, v) & \text{if } dom(a) \rightsquigarrow v \\ purge(\alpha, v) & \text{otherwise} \end{cases}
\end{aligned}
$$

- In other words, $purge(\alpha, v)$ deletes from α all actions from domains that are not allowed to interfere with domain v.

Secure System

- Informally, the system is secure if a given domain v is unable to distinguish between the state of the system after it has processed a given action sequence, and the state after processing the same sequence purged of actions required to be non-interfering with v.

Definition (Security)

A system is secure if for all $\alpha \in A^*$ and $a \in A$:

$$test(\alpha, a) = test(purge(\alpha, dom(a)), a)$$

- Abbreviation: $test(\alpha, a) = output(do(\alpha), a)$. We can think of the presentation of an action a and the observation of the output as an experiment performed by $dom(a)$ to find out something about the action sequence α.
- If $dom(a)$ can distinguish between the action sequences α and $purge(\alpha, dom(a))$ by such experiments then an action by some domain $u \not\rightsquigarrow dom(a)$ has "interfered" with $dom(a)$ and the system is insecure with respect to the policy $\rightsquigarrow$.

Holly, Lucy and the 2-Bit Machine

Example (Insecurity of the 2-bit machine)

Security-related definitions

- Security domains $D = \{H, L\}$ with $dom = \{((\text{Lucy}, \_), L), ((\text{Holly}, \_), H)\}$.
- Security policy $\rightsquigarrow = \{(L, H)\} \cup Id$, i.e., $\not\rightsquigarrow = \{(H, L)\}$.

Testing for (in)security

- Consider the sequence $\alpha = (\text{Holly}, \text{skip})(\text{Lucy}, \text{flip})(\text{Holly}, \text{flip})$.
- $purge(\alpha, L) = (\text{Lucy}, \text{flip})$, whereas $purge(\alpha, H) = \alpha$.
- $do(\alpha) = run(s_0, \alpha) = (0, 1)$ for initial state $s_0 = (0, 1)$.
- $do(purge(\alpha, L)) = (1, 0)$.
- $test(\alpha, (\text{Lucy}, \text{skip})) = 1 \neq 0 = test(purge(\alpha, L), (\text{Lucy}, \text{skip}))$.
- Therefore, M is insecure for $\rightsquigarrow$. Intuitively clear, since command (Holly, flip) from H domain influences L domain.

Holly, Lucy and the 2-Bit Machine

Example (Modified 2-bit machine M')

Modification

- $step((h, l), (\text{Holly}, \text{flip})) = (\overline{h}, l)$, i.e., flip only h bit, but not l bit.
- $step((h, l), (\text{Lucy}, \text{flip})) = (\overline{h}, \overline{l})$, and rest as before.

Repeating the previous test

- For $\alpha = (\text{Holly}, \text{skip})(\text{Lucy}, \text{flip})(\text{Holly}, \text{flip})$ as before we now have:
- $do(\alpha) = run(s_0, \alpha) = (0, 0)$ for initial state $s_0 = (0, 1)$.
- $do(purge(\alpha, L)) = (1, 0)$.
- $test(\alpha, (\text{Lucy}, \text{skip})) = 0 = test(purge(\alpha, L), (\text{Lucy}, \text{skip}))$.

Questions

- So far so good, but is M' now secure for $\rightsquigarrow$? Intuitively, it should be secure, since the influence of domain H on domain L is gone.
- How can we prove this?

Verification of Security

- The noninterference definition is expressed in terms of sequences of actions and state transitions;
- For the verification of the security of systems, we would like to derive local conditions on individual transitions.
- First step: partition the states of the system into equivalence classes that all "appear identical" to a given domain.
- The verification technique will then be to prove that each domain's view of the system is unaffected by the actions of domains that are required to be noninterfering with it.

Definition

A system M is view-partitioned if, for each domain $u \in D$, there is an equivalence relation $\overset{u}{\sim}$ on S. These equivalence relations are said to be output consistent if

$$s \overset{dom(a)}{\sim} t \;\Rightarrow\; output(s,a) = output(t,a).$$

Verification of Security

Lemma 1

Let $\rightsquigarrow$ be a policy and M a view-partitioned, output consistent system such that

$$do(\alpha) \overset{u}{\sim} do(purge(\alpha, u))$$

Then M is secure for $\rightsquigarrow$.

Proof.

Set $u = dom(a)$ and apply output consistency.

□

The Local Conditions

Definition

Let M be a view-partitioned system and $\rightsquigarrow$ a policy. We say that

- M locally respects $\rightsquigarrow$ if

$$dom(a) \not\rightsquigarrow u \Rightarrow s \overset{u}{\sim} step(s, a),$$

- M is step consistent if

$$s \overset{u}{\sim} t \Rightarrow step(s, a) \overset{u}{\sim} step(t, a).$$

- M locally respects $\rightsquigarrow$: Executing an action a from a domain not interfering with u yields a $\overset{u}{\sim}$-equivalent successor state.
- Step consistency: Executing the same action in each of two $\overset{u}{\sim}$-equivalent states again yields a pair of $\overset{u}{\sim}$-equivalent states.

The Unwinding Theorem

Theorem (Unwinding Theorem)

Let $\rightsquigarrow$ be a policy and M a view-partitioned system that is

1. *output consistent,*
2. *step consistent, and*
3. *locally respects $\rightsquigarrow$.*

Then M is secure for $\rightsquigarrow$.

Proof (Sketch).

We prove the following statement by induction on α:

$$P(\alpha) = \forall s, t.\ s \overset{u}{\sim} t \ \Rightarrow\ run(s, \alpha) \overset{u}{\sim} run(t, purge(\alpha, u))$$

The base case is elementary. In the inductive case, we have to show $P(a \cdot \alpha)$ assuming $P(\alpha)$. We do this by case analysis on $dom(a) \rightsquigarrow u$. Then the security of M for $\rightsquigarrow$ follows by setting $s = t = s_0$ and invoking Lemma 1. □

Setup for Access Control

Structured state A machine has a structured state if there exist

- a set N of names,
- a set V of values,
- $contents : S \times N \rightarrow V$

with the interpretation that $contents(s, n)$ is the value of the object named n in state s.

Access control

- $readable : D \rightarrow \mathcal{P}(N)$
- $writeable : D \rightarrow \mathcal{P}(N)$

with the interpretation that $readable(u)$/$writeable(u)$ denotes the objects that can be read/written by domain u.

Reference Monitor Assumptions (1)

First, we define, for $u \in D$, the relation $\overset{u}{\sim}$ on states by

$$s \overset{u}{\sim} t \text{ iff } (\forall n \in \mathit{readable}(u).\ \mathit{contents}(s, n) = \mathit{contents}(t, n))$$

Assumption (RMA 1)

We require that the output of an action a only depends on the values of objects to which $\mathit{dom}(a)$ has read access:

$$s \overset{\mathit{dom}(a)}{\sim} t \Rightarrow \mathit{output}(s, a) = \mathit{output}(t, a)$$

- This is, of course, output consistency, but note the concrete interpretation.

Reference Monitor Assumptions (2)

Recall: for $u \in D$, the relation $\overset{u}{\sim}$ on states is defined by

$$s \overset{u}{\sim} t \text{ iff } (\forall n \in \mathit{readable}(u).\ \mathit{contents}(s, n) = \mathit{contents}(t, n))$$

Assumption (RMA 2)

When an action a changes the system state, the new values of all changed objects must only depend on the values of objects to which $\mathit{dom}(a)$ has read access:

$$\begin{aligned} s \overset{\mathit{dom}(a)}{\sim} t \ \wedge\ & (\mathit{contents}(\mathit{step}(s, a), n) \neq \mathit{contents}(s, n) \\ & \vee\ \mathit{contents}(\mathit{step}(t, a), n) \neq \mathit{contents}(t, n)) \\ \Rightarrow\ & \mathit{contents}(\mathit{step}(s, a), n) = \mathit{contents}(\mathit{step}(t, a), n) \end{aligned}$$

- Second conjunct covers case, where n is not readable by $\mathit{dom}(a)$.
- In that case $\mathit{contents}(\mathit{step}(s, a), n) \neq \mathit{contents}(\mathit{step}(t, a), n)$ is legitimate.

Reference Monitor Assumptions (3)

Assumption (RMA 3)

If an action a changes the value of object n, then $dom(a)$ must have read access to n:

$$contents(step(s, a), n) \neq contents(s, n) \Rightarrow n \in writeable(dom(a))$$

General remarks on RMAs

- These requirements are standard for access control mechanisms.
- They capture the assumptions on the reference monitor that performs the access control function in any concrete instantiation of the theory.
- In particular, note that they are independent of any particular policy.

Access Control Theorem

Theorem (Access Control)

Let M be a system with structured state that satisfies the Reference Monitor Assumptions and the following two conditions:

1. $u \rightsquigarrow v \Rightarrow \mathit{readable}(u) \subseteq \mathit{readable}(v)$
2. $n \in \mathit{writable}(u) \wedge n \in \mathit{readable}(v) \Rightarrow u \rightsquigarrow v$

Then M is secure for $\rightsquigarrow$.

Proof (sketch).

It is not difficult to show that the conditions of the theorem satisfy those of the Unwinding Theorem.

- Output consistency: Immediate by first RMA.
- Step consistency: By case analysis on $\mathit{contents}(\mathit{step}(s,a),n) = \mathit{contents}(s,n) \wedge \mathit{contents}(\mathit{step}(t,a),n) = \mathit{contents}(t,n)$.
- Locally respects $\rightsquigarrow$: By contraposition $s \overset{u}{\not\sim} \mathit{step}(s,a) \Rightarrow \mathit{dom}(a) \rightsquigarrow u$.

□

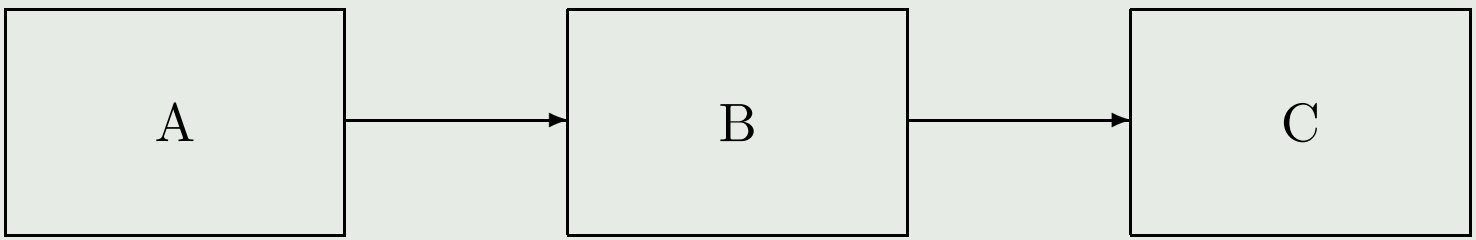
Outline

Transitive vs Intransitive

Transitive policies

- Only restriction so far: policy $\rightsquigarrow$ is reflexive.
- However, only transitive policies have a useful interpretation in current formulation of non-interference.

Example (Assured Pipeline)



Intended intransitive policy:

$$A \rightsquigarrow B,\ B \rightsquigarrow C,\ A \not\rightsquigarrow C$$

Problem: The current formulation of non-interference disallows *any* flow of information from A to C including information flowing through B.

Multi-level Security

Definitions

- Let L be a set of security labels with a partial ordering $\preceq$.
  Read $l_1 \preceq l_2$ as "l_1 is dominated by l_2".
- Let $clearance : D \to L$ be a function that assigns a security label to each domain.
- The associated multi-level security policy is

$$u \rightsquigarrow v \text{ iff } clearance(u) \preceq clearance(v) \tag{1}$$

- An arbitrary policy $\rightsquigarrow$ is MLS-type iff there is a label set L with a partial ordering $\preceq$ and a function $clearance : D \to L$ such that (1) holds.

Lemma

A policy is MLS-type if and only if it is transitive.

Bell-LaPadula Interpretation

Bell-LaPadula

- Assign labels to objects: $classification : N \to L$.
- ss-property: only read-down allowed.
- *-property: only write-up allowed.

Corollary (MLS)

Let $\rightsquigarrow$ be an MLS-type policy, and M a system with structured state that satisfies the Reference Monitor Assumptions and the following properties

ss-property: $n \in readable(u) \Leftrightarrow classification(n) \preceq clearance(u)$,

***-property:** $n \in writeable(u) \Rightarrow clearance(u) \preceq classification(n)$.

Then M is secure for $\rightsquigarrow$.

Proof.

Show that the two conditions mentioned in the statement of the Acess Control Theorem are satisfied. □

Completeness Result

Theorem (Completeness)

If M is a secure system for the policy $\rightsquigarrow$, then for each domain $u \in D$ there is an equivalence relation $\overset{u}{\sim}$ on the set of states such that the conditions of the unwinding theorem are satisfied.

- Note that the Theorem does not require that $\rightsquigarrow$ is transitive.

Proof (Sketch).

Define the relation $\overset{u}{\sim}$ as follows:

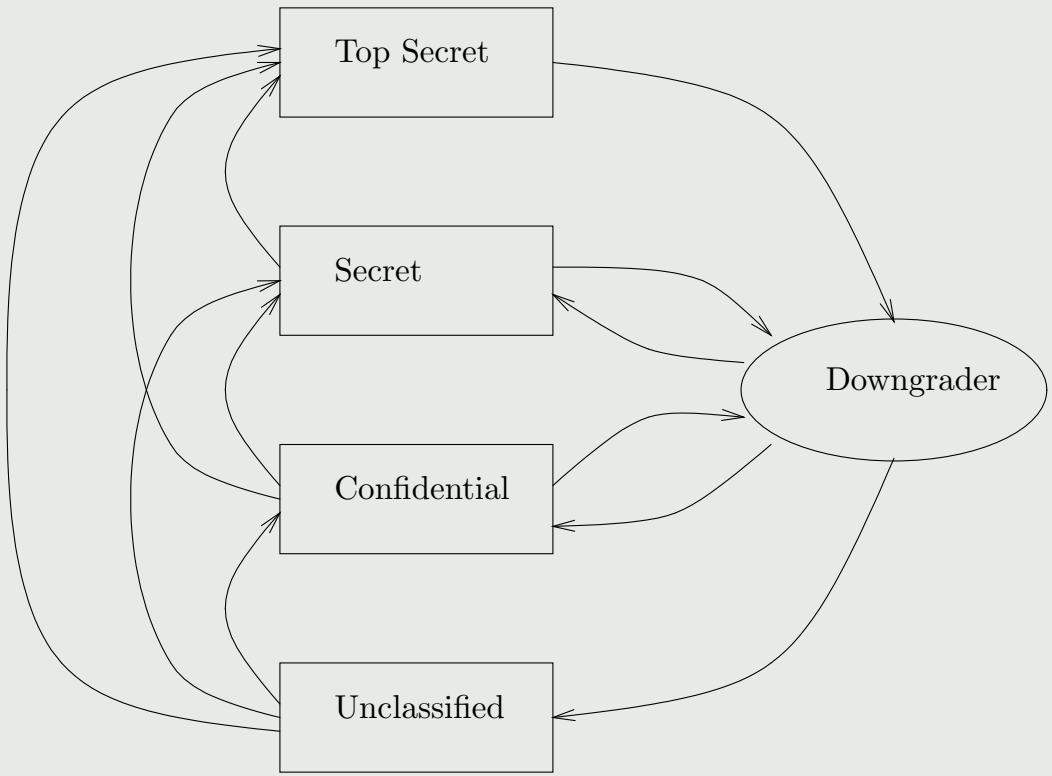
$$s \overset{u}{\sim} t \quad \text{iff} \quad (\forall \alpha \in A^*, b \in A.\ dom(b) = u \Rightarrow output(run(s, \alpha), b) = output(run(t, \alpha), b)).$$

This is clearly an equivalence relation. It remains to show that the three conditions of the unwinding theorem are satisfied. □

Outline

Transitive vs Intransitive (cont)

Example (MLS with downgrader)



- Left-hand side: standard transitive MLS setup (read down, write up).
- Right-hand side: trusted downgrader requires intransitive policy to mediate flows in opposite direction.

Intransitive Non-interference

Problem with current definition

- Current definition of non-interference too draconian.
- If $u \not\rightsquigarrow v$, the requirement is that deleting *all* actions performed by u should result in no change in the behaviour of the system as perceived by v.
- This is too strong if we also have the assertions $u \rightsquigarrow w$ and $w \rightsquigarrow v$, since it prohibits information flowing through w to influence v's view of the system.

Towards an improved definition

- Instead of deleting all actions performed by u, we should only delete those actions of u that are not followed by actions of w.
- For a formal definition we need to identify those actions in an action sequence that should not be deleted.

Legitimate Sources of Interference

Definition (Function *src*)

We define the function $src : A^* \times D \to \mathcal{P}(D)$ by the equations

$$src(\epsilon, u) = \{u\}$$

$$src(a \cdot \alpha, u) = \begin{cases} src(\alpha, u) \cup \{dom(a)\} & \text{if } \exists v.\, dom(a) \rightsquigarrow v \wedge v \in src(\alpha, u) \\ src(\alpha, u) & \text{otherwise} \end{cases}$$

$u \in src(\alpha, v)$ expresses that there exists a chain of interfering domains linking u to v and associated with a subsequence of α, i.e., we have

- $u = v$, or
- there is a subsequence of α consisting of actions by domains $w_1, w_2, \ldots, w_n$ such that $u = w_1 \rightsquigarrow w_2 \rightsquigarrow \cdots \rightsquigarrow w_n = v$.

For transitive $\rightsquigarrow$ we have $u \in src(\alpha, v)$ implies $u \rightsquigarrow v$.

Intransitive Purge

Definition (Function *ipurge*)

We define the function $ipurge : A^* \times D \to A^*$ by the equations

$$ipurge(\epsilon, v) = \epsilon$$

$$ipurge(a \cdot \alpha, v) = \begin{cases} a \cdot ipurge(\alpha, v) & \text{if } dom(a) \in src(a \cdot \alpha, v) \\ ipurge(\alpha, v) & \text{otherwise} \end{cases}$$

Example

Let $A = \{a, b, c\}$, $D = \{H, M, L\}$, $dom = \{(a, H), (b, M), (c, L)\}$ and $\rightsquigarrow = \{(H, M), (M, L)\} \cup Id$. Hence, $H \not\rightsquigarrow L$ and we have

$$\begin{array}{rcl@{\qquad}rcl} ipurge(\epsilon, L) & = & \epsilon & src(\epsilon, L) & = & \{L\} \\ ipurge(c, L) & = & c & dom(c) \in src(c, L) & = & \{L\} \\ ipurge(ac, L) & = & c & dom(a) \notin src(ac, L) & = & \{L\} \\ ipurge(bac, L) & = & bc & dom(b) \in src(bac, L) & = & \{M, L\} \\ ipurge(abac, L) & = & abc & dom(a) \in src(abac, L) & = & \{H, M, L\} \end{array}$$

Intransitive Security

Definition (Security)

A system M is secure for the policy $\rightsquigarrow$ if

$$test(\alpha, a) = test(ipurge(\alpha, dom(a)), a)$$

Lemma 2

Let $\rightsquigarrow$ be a policy and M a view-partitioned, output consistent system such that

$$do(\alpha) \overset{u}{\sim} do(ipurge(\alpha, u))$$

Then M is secure for $\rightsquigarrow$.

Unwinding Theorem

Definition (Weak step consistency)

Let M be a view-partitioned system and $\rightsquigarrow$ a policy. We say that M is weakly step consistent if

$$s \overset{u}{\sim} t \wedge s \overset{dom(a)}{\sim} t \Rightarrow step(s,a) \overset{u}{\sim} step(t,a)$$

- Cf. step consistency: Preservation of $\overset{u}{\sim}$ by an action a is only required if the states are also equivalent in $dom(a)$.

Theorem (Unwinding for intransitive non-interference)

Let $\rightsquigarrow$ be a policy and M a view-partitioned system that is

1. *output consistent,*
2. *weakly step consistent, and*
3. *locally respects $\rightsquigarrow$.*

Then M is secure for $\rightsquigarrow$.

Access Control Theorem

Theorem (Access control for intransitive non-interference)

Let M be a system with structured state that satisfies the Reference Monitor Assumptions and the condition

$$n \in \mathit{writable}(u) \wedge n \in \mathit{readable}(v) \;\Rightarrow\; u \rightsquigarrow v$$

Then M is secure for $\rightsquigarrow$.

- The first condition of the previous AC Theorem is no longer needed:

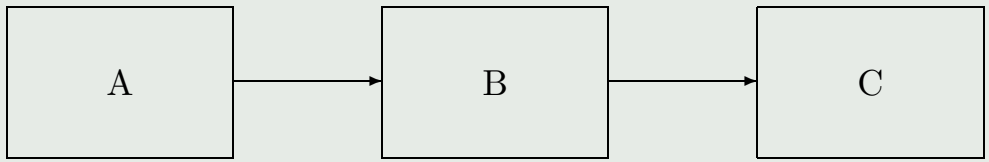
$$u \rightsquigarrow v \;\Rightarrow\; \mathit{readable}(u) \subseteq \mathit{readable}(v)$$

- This is because the Unwinding Theorem for the intransitive case, from which this theorem is derived, requires only weak step consistency.

Access Control Theorem

Discussion

Example (Assured Pipeline)



Intended intransitive policy: $A \rightsquigarrow B,\ B \rightsquigarrow C,\ A \not\rightsquigarrow C$.

- Intransitive AC Theorem allows A to write to an object n which B may read.
- Similarly, it permits B to write to an object m which C may read.
- In this way, information can flow from A to B and from B to C.
- However, A may not have write access to any objects to which C has read access; this prevents direct flow of information from A to C.
- Transitive AC Theorem additionally requires $\mathit{readable}(A) \subseteq \mathit{readable}(B)$ and $\mathit{readable}(B) \subseteq \mathit{readable}(C)$, and hence also $\mathit{readable}(A) \subseteq \mathit{readable}(C)$, i.e., A cannot hide any secrets from C. In example: C may read n himself.
- Thus, the additional condition of the Trans. AC Theorem forces the transitive completion of $\rightsquigarrow$ and so allows the direct flow of information from A to C.

Conclusions: Issues

Non-interference, and more generally, information flow security is a very active research area.

Many open questions

- What are appropriate formalizations for realistic systems? (e.g., infinite computations, non-determinism, probabilities)
- How do you prove these properties hold?
- Non-interference appears very strong. A secret password will influence system output, e.g., either login succeeds or fails!
- How to declassify certain kinds of information to allow interference in controlled ways?
- Mechanism design?

Conclusions: Extensions and Summary

Possible extensions

- Include policies that are not static, but dynamic (e.g., an access control matrix that changes over time).
- Non-deterministic systems.
- Probabilistic systems.

Summary

- Non-interference is an alternative, abstract formulation of security policy models.
- It states that a strict separation between security domains requires that all channels, no merely those designed to transmit information must be “closed”.

Bibliography

- Matt Bishop. *Computer Security (Art and Science).* Pearson, 2003.
- John McLean. *Security Models*, in J. Marciniak ed., Encyclopedia of Software Engineering, Wiley, 1994.
- J.A. Goguen and J. Meseguer. *Security Policies and Security Models.* Proceedings of Symposium on Privacy and Security, 1982.
- John Rushby. *Noninterference, transitivity, and channel-control security policies.* Technical Report CSL-92-02. Computer Science Laboratory, SRI International, Menlo Park, CA, 1992.
- Butler Lampson. *A note on the Confinement Problem*, 1973
- Gustavus J. Simmons. *Subliminal Communication is Easy Using the DSA*. 1993.

This module is mainly based on John Rushby's (very well written) paper.