Distributed Systems - Security
Foundations, Covert Channels, Non Interference

Marcus Völp / Hermann Härtig
2006 / 07
Purpose of this Lecture

- Formally define security
- Formally reason about security
- Security Evaluation (EAL 7 / A1 and beyond)
Overview

- Introduction
- Safety Question
  - Decidability and Protection Models
- Security Policies
  - Policy Enforcement
- Information Flow
  - Non Interference
Introduction: Security Policies

- **Definition:**
  - A *security policy* is a statement that partitions the states of the system into a set of authorized, or secure, states and a set of unauthorized, or nonsecure, states.
  - A *secure system* is a system that starts in an authorized state and cannot enter an unauthorized state.

- **Example:**
  - Policy: only root and I are allowed to read foo.txt
  - Enforcement: foo.txt u+r (g,a -r)
  - Secure system? No – owner can change rights to a+r
Introduction:
Confidentiality, Integrity, Availability

- Confidentiality:
  - Prevent unauthorized disclosure of information
    
    **Definition 1a:** Information $I$ is **confidential** with respect to set of entities $X$ if no member of $X$ can obtain information about $I$.

    **Definition 1b:** Only authorized users (entities, principals, etc.) can access information (data, programs, etc.)
Introduction: Confidentiality, Integrity, Availability

- **Integrity:**
  - Correctness of data and information (trust)
  
  *Definition 2a:* Information $I$ is **integer** with respect to $X$ if all members of $X$ trust $I$.

  *Definition 2b:* Either information is current, correct, and complete, or it is possible to detect that these properties do not hold.

- **Recoverability:**
  
  *Definition 3b:* Information that has been damaged can be recovered eventually.
Availability:

- Accessibility of information and services

**Definition 4a:** Resource $I$ is *available* with respect to $X$ if all members of $X$ can access $I$.

**Definition 4b:** Data is *available* when and where an authorized user needs it.
Introduction: Access Control Matrix

<table>
<thead>
<tr>
<th>Objects</th>
<th>File 1</th>
<th>File 2</th>
<th>Process1</th>
<th>Process2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subjects</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process1</td>
<td>read, write</td>
<td>read</td>
<td>read, write, execute</td>
<td>write</td>
</tr>
<tr>
<td>Process2</td>
<td>read</td>
<td>read</td>
<td>read</td>
<td>read, write, execute</td>
</tr>
</tbody>
</table>

- **Protection State Transitions:**
  - $X_i |- t_{i+1} X_{i+1}$  States $X_j$, Commands $t_k$
  - $X |- *Y$  Sequence
  - Access Control Matrix: $(S, O, P)$ with Subjects $S$, Objects $O$ and Permissions $P$
Introduction: Access Control Matrix

- **Commands**
  - **create subject s**
    - **Pre:** \( s \notin S, \)
    - **Post:** \( S' = S \cup \{s\}, \ O' = O \cup \{s\}, \)
    - \( \forall x \in O': p'(s, x) = \emptyset, \ \forall y \in S': p'(y, s) = \emptyset, \)
    - \( \forall x \in O, y \in S : p'(x, y) = p(x, y) \)
  
  - **enter r into p(s,o)**
    - **Pre:** \( s \in S, \ o \in O \)
    - **Post:** \( S' = S, \ O' = O, \)
    - \( \forall x \in O', y \in S': (s,o) \neq (x, y) \Rightarrow p'(x,y) = p(x, y) \)
    - \( p'(s, o) = p(s, o) \cup \{r\} \)
Introduction: Access Control Matrix

- Further operations:
  - create object \( o \)
  - delete right \( r \) from \( p(s,o) \)
  - destroy subject \( s \)
  - destroy object \( o \)
Principle of Attenuation

- A subject may not give rights it does not possess to another.

- **enter r into p(s,o)**
  - Pre: \( s \in S, o \in O \)
  - Post: \( S' = S, O' = O, \forall x \in O', y \in S': (s,o) \neq (x, y) \Rightarrow p'(x,y) = p(x, y) \)
  - \( p'(s, o) = p(s, o) \cup \{r\} \)

- **f.grant r into p(s,o)**
  - if \( r \) in \( p(f,o) \) then
  - **enter r into p(s,o)**
Safety Question

- **Definition: Leakage**
  When a right \( r \) is added to an element of the ACM not already containing \( r \), \( r \) is said to be leaked.

- Is the system *safe with respect to right* \( r \), i.e., can it never happen that the system (including \( s_0 \)) leaks the right \( r \)?

- **Safety Question:**
  Is there an algorithm for determining whether a given protection system with initial state \( s_0 \) is safe with respect to \( r \)?
Safety Question: Decidability

**Theorem:**
It is undecidable whether a given state of a given protection system is safe for a given generic right.

**Proof by contradiction:**
Reduction of the halting problem of an arbitrary Turing machine to the safety problem. (next slide)

However, safety is decidable systems with more specific rules:
- Monoconditional (only one condition in if clause) monotonic (no destroy command) systems.
- Take-Grant protection model
Safety Question: Decidability

Proof Sketch:

- Turing Machine: \( T \) (tape symbols \( M \), states \( K \), \( \delta \))
  - \( \delta: K \times M \rightarrow K \times M \times \{L,R\} \)
    - e.g., \( \delta: (x, A) \rightarrow (y, B, L) \)

- "Implement Turing Machine with ACM"
  - states, symbols \( \rightarrow \) generic rights
  - cell \( i \) \( \rightarrow \) subject \( s_i \)
  - Head:
    - head in cell \( j \), \( T \) in state \( x \) \( \Rightarrow \) \( x \in p(s_j, s_j) \)
Safety Question: Decidability

- **Proof Sketch:**

  - **Command** $\delta$: $(x, A) \rightarrow (y, B, L)$
    - if own in $p(s_{i-1}, s_i)$ and $x$ in $p(s_i, s_i)$ and $A$ in $p(s_i, s_i)$ then
      - delete $x$ from $p(s_i, s_i)$
      - delete $A$ from $p(s_i, s_i)$
      - enter $B$ into $p(s_i, s_i)$
      - enter $y$ into $p(s_{i-1}, s_{i-1})$
    - Similar commands for other $\delta$

  If Turing machine enters state $q_f$ then the protection system has leaked right $q_f$; otherwise the protection system is safe for generic right. But whether $T$ enters the (halting) state $q_f$ is undecidable.
Take-Grant Protection Model

- Directed Graph
  - Vertices: object, subject (either object or subject)
  - Edges: subject has right $r$ on object

- Transition Rules:
  - Take
  ![Take Transition Diagram]
  - Grant
  ![Grant Transition Diagram]
  - Create
  ![Create Transition Diagram]
  - Remove
Take-Grant Protection Model

- Sharing and Thieves
  - can share ($\alpha, x, z, G_0$)
  - Lemma:
    - Proof:
      $$x\.create v \ (tg) \ ; \ y\.take g \ ; \ y\.grant \ \alpha \ \text{to} \ v \ ; \ x\.take \ \alpha \ \text{from} \ v$$

- can steal ($\alpha, x, z, G_0$)
  - No owner of right $\alpha$ will grant rights to another.
Take-Grant Protection Model

- Decidability: Islands and Bridges
  - TG-Path: sequence of vertices $v_0, \ldots, v_n$ with edge (in either direction) $v_i, v_{i+1}$ containing $t$ or $g$

- Island: maximal tg-connected subject-only subgraph

- Bridge: tg-path with subject endpoints and edges associated words in:
  $$\{t^*, t^*g, t^*t, t^*gt^*\}.$$

- can share $(\alpha, x, z, G_0) \iff$
  - $s$ in $G_0$ with $s \xrightarrow{\alpha} y$
  - subject $x'$ with $x' = x$ or tg-path $x' \rightarrow s$ with $\{t^* g\}$
  - subject $s'$ with $s' = s$ or tg-path $s' \rightarrow s$ with $\{t^*\}$
  - Islands $I_1 \ldots I_n$, $x'$ in $I_1$, $s'$ in $I_n$, bridge $I_j \rightarrow I_{j+1}$
Summary

- Security is concerned with
  - Confidentiality
  - Integrity
  - Availability

- Safety
  - Not decidable in general
  - Not decidable for unrestricted Access Control Matrix
  - There are decidable protection models
    (e.g., Take-Grant Capability Model)
Overview

- Introduction
- Safety Question
  - Decidability and Protection Models
- Security Policies
  - Policy Enforcement
- Information Flow
  - Covert Channels
    - Definition
    - Detection
  - Non Interference and Unwinding Theorems
Security Policies

- Classification
  - Concern:
    - Confidentiality Policies  e.g., Bell La Padula
    - Integrity Policies  e.g., Biba, (Inventory System)
    - Availability Policies
    - Hybrid  e.g., Chinese Wall,  (Clinical Information System)

- Discretionary
  - User can set access control mechanism to allow or deny access to an object.

- Mandatory
  - System mechanism controls access to an object; individual users cannot alter this access.
Multi Level Security

- ***-property**
  - S can write O if and only if Label(S) <= Label(O)

- **basic security condition**
  - S can read O if and only if Label(O) <= Label(S)

Relation ≤ : L x L defines total order of labels
Lattice [D. Denning '76]

- Relation \( \leq \) defines partial order of security levels
- Least upper bound exists for any finite subset

Confidentiality: \( L \leq H \)
Integrity: \( h \leq l \)
Low-Water-Mark / Biba Integrity Policy

- Integrity Labels similar to secrecy labels:
  - Idea: Data produced by source of varying trusted.
  - Using less trusted data will influence the results

- Low Water Mark
  - s can write to o if and only if I(o) <= I(s)
  - If s reads o then I'(s) = min(I(s), I(o))
  - s₁ can execute s₂ if and only if I(s₂) <= I(s₁)
    - Problem: decrease of integrity level

- Biba
  - s can read o if and only if I(s) <= I(o)
  - s can write o if and only if I(o) <= I(s)
  - s₁ can execute s₂ if and only if I(s₂) <= I(s₁)
Chinese Wall

- **Conflict of Interest**
  - British law e.g., in stock exchange
    - Trader represents two clients and best interest of clients conflict (trader could help one gain at expense of other)

  **Conflict of interest classes**

- **Simple Security**
  - $S$ can read $O$ iff
    - exists $O'$ accessed by $S$ with $CD(O') = CD(O)$, or,
    - Forall $O'$ read by $S$ => $COI(O') \neq COI(O)$

  - * property
    - $S$ may write $O$ iff
      - $S$ can read $O$, and,
      - Forall $O'$ readable by $S$ => $CD(O') = CD(O)$
Policy Enforcement Mechanisms

- **Access Control List (classical)**
  - OS keeps list of processes x rights for each object
  - $\text{acl}(\text{file1}) = \{(\text{process 1, \{read, write, execute\}}), (\text{process 2 \{read\}})\}$
  - $\text{acl}(\text{process1}) = \{(\text{process 1, \{read, write, execute\}})\}$
  - $\text{acl}(\text{process2}) = \{(\text{process 1, \{write\}}), (\text{process 2, \{r, w, x\}})\}$

- **Abbreviations:**
  - Groups: Unix, AIX
  - Wildcards:
    - $p, *, \text{read}$ (read access to $p$ regardless in which group $p$ is)
  - **Conflicts:**
    - two opposing rights in ACL (group +r, user –r)
      - order of occurrence in ACL: Cisco Router
      - deny > allow: AIX
  - **Problems:** modification, revocation
Policy Enforcement Mechanisms

- Capabilities
  - `caps(process 1) = {(file1, {read, write}), (file2, {read})}`

- Implementation:
  - Store capabilities in per process segment / page protected by kernel (e.g. page permission = supervisor) (e.g., CAP)
  - Cryptography (e.g., Amoeba)
  - Hardware tags associated with each word (rarely used e.g., B5700)

- Copying:
  - Take, grant permissions on capabilities
  - Copy flag

- Revocation:
  - Local:
    - Linked list / Tree (e.g., Mapping Database) of all capabilities
    - Indirection: Object which stores capabilities, indirection right authorizes use but not take or grant of capability
      revoke by destroying indirection object
  - Remote:
    - Expiry information
Policy Enforcement Mechanisms

- Monitoring: (Schneider / Bauer)

Each operation of A generates an input into security automaton of monitor

If monitor can make transition, operation of A is authorized.
If not, the monitor stops A before B sees the result.

Bauer: the automaton can edit the results
Enforceable Security Policies
Overview

- Introduction
- Safety Question
  - Decidability and Protection Models
- Security Policies
  - Policy Enforcement
- Information Flow
  - Non Interference
Information Flow

- Information Flow Policies
  - Bell La Padua
  - Lattice Security
  - Chinese Wall

transitive flows

intransitive flows

high

low

encrypt

low

low
Information Flow

- Intuitively: information flows from object x to object y if the application of command sequence c causes information initially in x to affect information in y.

- Information Theory
  - Entropy: $H(X) = - \sum p(X=x_i) \log_2 p(X=x_i)$

  - The command sequence c causes a flow of information from x to y if $H(x_s|y_t) < H(x_s|y_s)$.

    with x,y objects and s |- c t
Noninterference

- Motivation:
  - Single property to express and reason about security policies
    - Confidentiality:
      - \( A \sim/\sim \rightarrow B \Rightarrow \)
      - B cannot deduce information on A (A’s data), A is confidential with respect to B
    - Integrity:
      - \( A \sim/\sim \rightarrow B \Rightarrow \)
      - B’s execution is independent of information / results from A, B is integer with respect to A
    - Availability:
      - \( A \sim/\sim \rightarrow B \Rightarrow \)
      - B’s availability is independent of information / results from A, B’s availability cannot be affected by A
Noninterference

- Intuitively:
  - Given two subsystems a, b:
    - a cannot interfere with b if outputs produced by a cannot be seen by b
  - Noninterference Relation: $a \not\rightarrow b$
Noninterference

- **Formal Definition:**
  - Security Domains
    - \( \text{dom: object} \rightarrow \text{domain} \)
  - Security Policy
    - \( d \rightarrow d' : \text{domain} \times \text{domain} \) information flow is allowed from \( d \) to \( d' \)
    - \( \sim/\sim> : \text{domain} \times \text{domain} \) (complement of \( \sim> \))
  - Helpers
    - Purge: remove all actions of domains \( u \) with \( u\sim/\sim>v \) (they must not affect \( v \))

\[
\text{purge}([],v) = []; \text{purge}(a^{\alpha}) = \begin{cases} 
  a^{\alpha}\text{purge}(\alpha) & \text{if dom(a) \( \rightarrow v \)} \\
  \text{purge}(\alpha) & \text{otherwise}
\end{cases}
\]

- **Noninterference**
  - \( \text{output(run(a, s), d)} = \text{output(run(purge(a, d), s), d)} \)
Noninterference - Unwinding

- Local properties for individual program (OS) steps \((I)\) that yield noninterference? \(\Rightarrow\) Unwinding Theorems

- Here only the results
  - define what can be observed by an \(I\)-classified observer:
    \[
    S_x \sim_I S_y
    \]
  - Noninterference \(\Leftrightarrow\)
    \[
    S_x \sim_I S_y \Rightarrow [I](S_x) \sim_I [I](S_y)
    \]
Examples: Confidentiality of Programs

```c
int l {low};  // variable that is externally observable after program terminates
int h {high}; // variable storing confidential data

void foo() {
    l = h;
}

void bar() {
    if (h % 2) == 1 {
        l = 1;
    }
}

void sec() {
    if (h % 2) == 1 {
        h = h + 4;
    }
}

void long_op() {
    if (h % 2) == 1 {
        while (int i < 10000) { i++; }
    }
}

void terminate() {
    if (h%2) == 1 {
        while (true);
    }
}
```
Secure Type System

- Program is noninterference secure if it is typeable
  - Notation:
    - $\vdash \text{exp} : t$ expression has type $t$ according to typing rules
    - $[\text{pc}] \vdash C$ programm $C$ is typeable in security context $[\text{pc}]$

- Typing rules for while language

\[
\begin{align*}
[E1-2] & \quad \vdash \text{exp} : \text{high} \quad \frac{h \notin Vars(\text{exp})}{\vdash \text{exp} : \text{low}} \\
[C1-3] & \quad [\text{pc}] \vdash \text{skip} \quad [\text{pc}] \vdash \text{h} := \text{exp} \quad \vdash \text{exp} : \text{low} \\
[C4-5] & \quad [\text{pc}] \vdash C_1 \quad [\text{pc}] \vdash C_2 \quad \frac{[\text{pc}] \vdash \text{exp} : \text{pc} \quad [\text{pc}] \vdash C}{[\text{pc}] \vdash \text{while exp do C}} \\
[C6-7] & \quad [\text{pc}] \vdash \text{exp} : \text{pc} \quad [\text{pc}] \vdash C_1 \quad [\text{pc}] \vdash C_2 \quad [\text{pc}] \vdash \text{if exp then } C_1 \text{ else } C_2 \quad [\text{pc}] \vdash \text{high} \vdash C \quad [\text{pc}] \vdash \text{low} \vdash C
\end{align*}
\]
Questions

- **References**
  - Matt Bishop: *Computer Security – Art and Science*
  - P. Gallagher: *A Guide to Understanding Covert Channel Analysis of Trusted Systems* [TCSEC]
  - Proctor, Neumann: *Architectural Implications of Covert Channels*
  - Kemmerer, Porras: *Covert Flow Trees: A visual approach to detecting covert storage channels*
  - Sabelfeld, Myers: *Language-based information-flow security*
  - Walker, Bauer, Ligatti: *More enforcable security policies*