Course Outline

1. Introduction: Role of security, Types of security, Definitions.
2. Access Control Matrix model
3. Protection Models
5. Formal policy models.
6. Information Flow
7. Authentication and Identity
8. Forensics

Midterm. 10/20

13. Malicious Code: Viruses, Worms, etc.
15. Physical threats, operational security, Legal and Societal Issues

Final Exam
Basic Components

- Confidentiality
  - Keeping data and resources hidden

- Integrity
  - Data integrity (integrity)
  - Origin integrity (authentication)

- Availability
  - Enabling access to data and resources

Policies and Mechanisms

- Policy says what is, and is not, allowed
  - This defines “security” for the site/system/etc.
  - Policy definition: Informal? Formal?

- Mechanisms enforce policies

- Composition of policies
  - If policies conflict, discrepancies may create security vulnerabilities
Access Control

- **State**: Status of the system
  - Protection state: subset that deals with protection
- **Access Control Matrix**: Describes protection state
- **Formally**:
  - Objects $O$
  - Subjects $S$
  - Matrix $A \subseteq S \times O$
- **Tuple** $(S, O, A)$ defines protection states of system

### Access Control Matrix: Boolean Evaluation Example

<table>
<thead>
<tr>
<th></th>
<th>Internal</th>
<th>Local</th>
<th>State University</th>
<th>Long Distance</th>
<th>International</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>CR</td>
<td>T</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Student</td>
<td>CR</td>
<td>T</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Staff</td>
<td>CR</td>
<td>Transfer</td>
<td>CR</td>
<td>T</td>
<td>R</td>
</tr>
<tr>
<td>Account</td>
<td>CR</td>
<td>T</td>
<td>CR</td>
<td>CR</td>
<td>CR T</td>
</tr>
</tbody>
</table>

© 2010 Chris Clifton
Protection State Transitions

- State $X_i = (S_i, O_i, A_i)$
- Transitions $\tau_i$
  - Single transition $X_i \vdash_{\tau_{i+1}} X_{i+1}$
  - Series of transitions $X \vdash^* Y$
- Access control matrix may change
  - Change command $c$ associated with transition
    $X_i \vdash_{c_{i+1}(p_{i+1}, \ldots, p_{i+1})} X_{i+1}$
- Change command $c$ associated with transition

Primitive Commands

- Create Object $o$
  - Adds $o$ to objects with no access
    $S' = S$, $O' = O \cup \{o\}$, $(\forall x \in S')[a'[x,o] = \emptyset]$, $(\forall x \in S')(\forall y \in O)[a'[x,y] = a[x,y]]$
- Create Subject $s$
  - Adds $s$ to objects, subjects, sets relevant access control to $\emptyset$
- Enter $r$ into $a[s,o]$
- Delete $r$ from $a[s,o]$
- Destroy subject $s$, destroy object $o$
Formally:

- Given
  - initial state $X_0 = (S_0, O_0, A_0)$
  - Set of primitive commands $c$
- Can we reach a state $X_n$ where $\exists s, o$ such that $A_n[s,o]$ includes a right $r$ not in $A_0[s,o]$?
  - If so, the system is not safe

Decidability Result
*(Harrison, Ruzzo, Ullman)*

- Given a system where each command consists of a single *primitive* command, There exists an algorithm that will determine if a protection system with initial state $X_0$ is safe with respect to right $r$.
- Proof: determine minimum commands $k$ to leak
  - Delete/destroy: Can’t leak (or be detected)
  - Create/enter: new subjects/objects “equal”, so treat all new subjects as one
  - If $n$ rights, leak possible, must be able to leak in $n(|S_0|+1)(|O_0|+1)+1$ commands
- Enumerate all possible to decide
Other Results
*(most from the same authors)*

- Set of unsafe systems recursively enumerable
- Without create primitive, safety in P-SPACE
  - Like halting problem reduction, but no unlimited tape
- Without delete/destroy, still undecidable
  - Decidable if at most one condition allowed per command
  - Still holds if delete allowed

Take-Grant Protection Model

- System is directed graph
  - Subject: ●
  - Object: ○
  - *(labeled)* edge: {rights}
- Take rule: if \( t \in \gamma, \alpha \subseteq \beta \), can add transitive edge
- Grant rule: if \( g \in \zeta, \alpha \subseteq \gamma \), can add (grant) edge between recipients
- Create, Remove rules
Take-Grant Protection Model: Sharing

- Given $G_0$, can vertex $x$ obtain $\alpha$ rights over $y$?
  - $\text{Can\_share}(\alpha, x, y, G_0)$ iff $G_0 \xrightarrow{\ast} G_n$, using the above rules and $\alpha$ edge from $x$ to $y$ in $G_n$
- $tg$-path: $v_0, \ldots, v_n$ where $t$ or $g$ edge between any $v_i, v_{i+1}$
  - Vertices $tg$-connected if $tg$-path between them
- Theorem: Any two subjects with $tg$-path of length 1 can share rights

---

Theorem: $\text{Can\_share}(\alpha, x, y, G_0)$

- $\text{Can\_share}(\alpha, x, y, G_0)$ iff there is an $\alpha$ edge from $x$ to $y$ in $G_0$ or if:
  - $\exists$ a vertex $s \in G_0$ with an $s$ to $y$ $\alpha$ edge,
  - $\exists$ a subject $x'$ such that $x' = x$ or $x'$ initially spans to $x$,
  - $\exists$ a subject $s'$ such that $s' = s$ or $s'$ terminally spans to $s$, and
  - $\exists$ islands $l_1, \ldots, l_n$ such that $x' \in l_i, s' \in l_n$, and there is a bridge from $l_i$ to $l_{i+1}$
- Proof: If: $x'$ grants to $x$, $s'$ takes from $s$, otherwise as with subjects
  - Only if: as before, plus object can’t give (receive) a right unless someone can take (grant) it
- Corollary: There is an $O(|V|+|E|)$ algorithm to test $\text{can\_share}$
Theorem: When Theft Possible

- Can_steal(α,x,y,G₀) iff there is no α edge from x to y in G₀ and ∃ G₁, ..., Gₙ s. t.:
  - There is no α edge from x to y in G₀,
  - ∃ subject x' such that x'=x or x' initially spans to x, and
  - ∃ s with α edge to y in G₀ and can_share(t,x',s,G₀)

- Proof:
  - ⇒: (easy – build path)
  - ⇐: Assume can_steal:
    - No α edge from definition.
    - Can_share(α,x,y,G₀) from definition: α from x to y in Gₙ
    - s exists from can_share and Monday's theorem
    - Can_share(t,x',s,G₀): s can't grant α (definition), someone else must get α from s, show that this can only be accomplished with take rule

Schematic Protection Model

- Key idea: Protection Type τ
  - Label that determines how control rights affect an entity
  - Take-Grant: subject and object are different protection types
  - Unix file system: File, Directory, ???

- Ticket: Describes a set of rights
  - Entity has set dom(X) of tickets Y/z describing X's rights z over entities Y

- Inert right vs. Control right
  - Inert right doesn’t affect protection state
Transferring Rights

- **Link predicate:** $\text{link}_i(X, Y)$
  - conjunction or disjunction of
    - $X/z \in \text{dom}(X)$, $X/z \in \text{dom}(Y)$
    - $Y/z \in \text{dom}(X)$, $Y/z \in \text{dom}(Y)$
    - $\text{true}$
  - Determines if $X$ and $Y$ “connected” to transfer right
  - Example: $\text{link}(X, Y) = Y/g \in \text{dom}(X) \lor X/t \in \text{dom}(Y)$

- **Filter function:** conditions on transfer
- Copy $X/r:c$ from $Y$ to $Z$ allowed iff $\exists i$ such that:
  - $X/rc \in \text{dom}(Y)$
  - $\text{link}_i(Y, Z)$
  - $\tau(X)/r:c \in \text{filter}_i(\tau(Y), \tau(Z))$

Safety Analysis in SPM

- **Idea:** derive *maximal state* where changes don’t affect analysis
  - Similar to determining max flow

- **Theorems:**
  - A maximal state exists for every system
  - If parent gives child only rights parent has (conditions somewhat more complex), can easily derive maximal state
Typed Access Matrix Model

- Finite set $T$ of types ($TS \subseteq T$ for subjects)
- Protection State: $(S, O, \tau, A)$
  - $\tau: O \rightarrow T$ is a type function
  - Operations same as Harrison-Ruzzo-Ullman except create adds type
- $\tau$ is child type iff command creates create subject/object of type $\tau$ (otherwise parent)
- If parent/child graph from all commands acyclic, then:
  - Safety is decidable
  - Safety is NP-Hard
  - Safety is polynomial if all commands limited to three parameters

Comparing Models

- Expressive Power
  - HRU/Access Control Matrix subsumes Take-Grant
  - HRU subsumes Typed Access Control Matrix
  - SPM subsumes Take-Grant
    - Subject/Object protection types
    - ticket is label on an edge
    - take/grant are control rights
- What about SPM and HRU?
  - SPM has no revocation (delete/destroy)
- HRU without delete/destroy (monotonic HRU)?
  - MTAM subsumes monotonic mono-operational HRU
  - HRU can have create requiring multiple “parents”
Extended Schematic Protection Model

- Adds “joint create”: new node has multiple parents
  - Allows more natural representation of sharing between mutually suspicious parties
    - Create joint node for sharing
  - In Take-Grant, SPM, must create two nodes, they interact to share (equivalent power)
- Monotonic ESPM and Monotonic HRU equivalent

Security Mechanism

- Policy describes what is allowed
- Mechanism enforces (part of) policy
  - The two need not be the same!
- Example Policy: Students should not copy homework
  - Mechanism: Disallow access to files owned by other users
- Does mechanism enforce policy?
  - Is mechanism too strict?
Modeling Secure/Precise: Confidentiality (Jones and Lipton)

- What are we modeling? A program
  - $p: I_1 \times \ldots \times I_n \rightarrow R$ is a program
  - Defined in terms of inputs and outputs
  - Goal: Determine if $p$ can violate confidentiality
- Observability
  - Output of function $p(i_1, \ldots, i_n)$ encodes all available information on inputs $i_1, \ldots, i_n$
  - Output may include things not normally thought of as part of function result
    - Data accessed
    - Timing
    - Anything that can be observed

Bell-LaPadula: Basics

- Mandatory access control (Security Level)
  - Subject has clearance $L(S) = l_s$
  - Object has classification $L(O) = l_o$
  - Clearance/Classification ordered
    - $l_i < l_{i+1}$
- Discretionary access control
  - Matrix: Subject has read (write) on Object
- Need both to perform operation
Access Rules

• Simple Security Condition: S can read O if and only if
  – S dom O and
  – S has discretionary read access to O
• *-Property: S can write O if and only if
  – O dom S and
  – S has discretionary write access to O
• Secure system: One with above properties
• Theorem: Let Σ be a system with secure initial state σ₀, T be a set of state transformations
  – If every element of T follows rules, every state σᵢ secure

Formalizing Bell-LaPadula

• Objects in a hierarchy h: O → P(O)
  – o_i ≠ o_j ⇒ h(o_i) ∩ h(o_j) = ∅ (no two nodes at same point)
  – There is no {o_1, o_2, ..., o_k} ⊆ O such that ∀ i = 1, ..., k, o_i+1 ∈ h(o_i) and o_k = o_1 (no cycles)
• State v ∈ V is a 4-tuple (b,m,f,h)
  – b ∈ P(S × O × P) indicates which subjects can access which objects and what the rights are
• R denotes requests for access
• D set of outcomes
  – yes, no, illegal, error
• Actions W ⊆ R × D × V × V
  – Request leads to outcome, moving from one state to another
• System Σ(R, D, W, z₀) ⊆ Rᴺ × Dᴺ × Vᴺ
  – Set of states that result from a given set of actions
  – (r,d,v,v') ∈ W an action of Σ iff ∃ time t, (x,y,z) ∈ Σ such that (r,d,v,v) = (xₜ,yₜ,zₜ,zₜ₋₁)
A System is Secure if it Satisfies:

- Simple security condition satisfied for \((s, o, p) \in S \times O \times P\) relative to \(f\) iff
  - \(p = e\) or \(p = a\)
  - \(p = r\) or \(p = w\) and \(f_\delta(s) \text{ dom } f_\delta(o)\)

- *-property satisfied for \((b, m, f, h)\) iff \(\forall s \in S\)
  - \(b(s:a) \neq \emptyset \Rightarrow [\forall o \in b(s: a) [f_\delta(o) \text{ dom } f_\delta(s)]]\)
  - \(b(s:w) \neq \emptyset \Rightarrow [\forall o \in b(s: w) [f_\delta(o) = f_\delta(s)]]\)
  - \(b(s:r) \neq \emptyset \Rightarrow [\forall o \in b(s: r) [f_\delta(s) \text{ dom } f_\delta(o)]]\)

- Discretionary security property satisfied for \((b, m, f, h)\) iff \(\forall (s, o, p) \in b, p \in m[s,o]\)

Modeling with Bell-LaPadula: get-read

- \(r = (\text{get}, s, o, r) \in R^{(1)}\) request
- \(v = (b, m, f, h)\) system state
- if \(r \notin \Delta(p_1)\) then \(p_1(r,v) = (i, v)\) bad arguments
  else if \((f_\delta(s) \text{ dom } f_\delta(o))\) ssc preserving
    and \([s \in S_T \text{ or } f_\delta(s) \text{ dom } f_\delta(o)]\)
    and \(r \in m[s,o]\) discretionary access control
    then \(p_1(r,v) = (y, (b \cup \{ (s, o, r) \}, m, f, h))\)
  else \(p_1(r,v) = (n, v)\)

- Theorem: get-read is secure
  - Assume \(v\) secure
  - Either \(v' = v\) or \(v' = v\) with \(\{ (s, o, r) \}\) added to accesses
    - \((s, o, r)\) must satisfy security properties to reach where it is added
- Similar rules for get-append, execute, write
Integrity Policy

- **Principles:**
  - Separation of Duty: Single person can’t mess up the system
    - No coding on live system
  - Separation of function
    - No development on production data
  - Auditing
    - Controlled/audited process for updating code on production system
- **This enables validated code to maintain integrity**
  - *But how do we ensure we’ve accomplished these?*
  - *Is this overkill?*

Policies

- **Ring Policy**
  - \( s \circ o \)
  - \( s \circ w \circ o \leftrightarrow i(o) \leq i(s) \)
  - \( s_1 \times s_2 \leftrightarrow i(s_2) \leq i(s_1) \)
- **Low-Water-Mark Policy**
  - \( s \circ o \Rightarrow i(s) = \min(i(s), i(o)) \)
  - \( s \circ w \circ o \leftrightarrow i(o) \leq i(s) \)
  - \( s_1 \times s_2 \leftrightarrow i(s_2) \leq i(s_1) \)
- **Biba’s Model: Strict Integrity Policy**
  - \( s \circ o \leftrightarrow i(s) \leq i(o) \)
  - \( s \circ w \circ o \leftrightarrow i(o) \leq i(s) \)
  - \( s_1 \times s_2 \leftrightarrow i(s_2) \leq i(s_1) \)
- **Theorem for induction similar to Bell-LaPadula**
Domain-specific Policy Models

- Military Confidentiality
  - Bell-LaPadula
- Database Integrity
  - Clark/Wilson
- Corporate Anti-Trust
  - Chinese Wall
- Clinical Information Systems
- Others?

What is Consistent?

- Principle of autonomy:
  - Access allowed by security policy of a component must be allowed by composition
- Principle of security:
  - Access denied by security policy of a component must be denied by composition
- Must prove new “composed” policy meets these principles
Information Flow

- Information Flow: Where information can move in the system
- How does this relate to confidentiality policy?
  - Confidentiality: What subjects can see what objects
  - Flow: Controls what subjects actually see
- Variable $x$ holds information classified $S$
  - $x$, information flow class of $x$, is $S$
- Confidentiality specifies what is allowed
- Information flow describes how this is enforced

Formal Definition

- Problem: capturing all information flow
  - Files
  - Memory
  - Page faults
  - CPU use
  - $?$
- Definition: Based on entropy
  - Flow from $x$ to $y$ (times $s$ to $t$) if $H(x_s | y_t) < H(x_s | y_s)$
How do we Manage Information Flow?

- Information flow policy
  - Captures security levels
  - Often based on confinement
  - Principles: Reflexivity, transitivity
- Compiler-based mechanisms
  - Track potential flow
  - Enforce legality of flows
- Execution-based mechanisms
  - Track flow at runtime
  - Validate correct

Confinement

- Confinement Problem
  - Prevent a server from leaking confidential information
- Covert Channel
  - Path of communication not designed as communication path
- Transitive Confinement
  - If a confined process invokes a second process, invokee must be as confined as invoker
Isolation

- Virtual machine
  - Simulates hardware of an (abstract?) machine
  - Process confined to virtual machine
    - Simulator ensures confinement to VM
  - Real example: IBM VM/SP
    - Each user gets “their own” IBM 370
- Sandbox
  - Environment where actions restricted to those allowed by policy

Covert Channels

- Storage channel
  - Uses attribute of shared resource
- Timing channel
  - Uses temporal/ordering relationship of access to shared resource
- Noise in covert channel
  - Noiseless: Resource only available to sender/receiver
  - Noisy: Other subjects can affect resource
Modeling Covert Channels

• Noninterference
  – Bell-LaPadula approach
  – All shared resources modeled as subjects/objects
  – Let $\sigma \in \Sigma$ be states. Noninterference secure if $\forall \sigma$ at level $l(\sigma) \ni \Sigma \times \Sigma$ such that
    • $\sigma_1 \equiv \sigma_2 \Rightarrow \text{view}(\sigma_1) = \text{view}(\sigma_2)$
    • $\sigma_1 \equiv \sigma_2 \Rightarrow \text{execution}(i, \sigma_1) \equiv \text{execution}(i, \sigma_2)$
    • if $i$ only contains instructions from subjects dominating $\sigma$, $\text{view}(\text{execution}(i, \sigma)) = \text{view}(\sigma)$

• Information Flow analysis
  – Again model all shared resources

Test Taking Hints

• Open book/notes
  – Pretty much any non-electronic aid allowed
• See old copies of my exams (and solutions) at my web site
  – CS 526
  – CS 541
  – CS 603
• Time will be tight
  – Suggested “time on question” provided