Advanced Systems Security: Principles

Trent Jaeger
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University
Access Control – The Right Way

- We said that ordinary operating systems cannot control code controlled by an adversary
- Review formalisms developed for “protection”
  - and show how they are extended to enforce “security”
- Key concepts
  - Mandatory protection state
    - Adversary cannot modify access control policy
  - Reference monitor
    - Enforce access control comprehensively
  - Later: Security models
Protection System

• Manages the authorization policy for a system
  ‣ It describes what operations each subject (via their processes) can perform on each object

• Consists of
  ‣ **State:** Protection state
  ‣ **State Ops:** Protection state operations
The Access Matrix

- An access matrix is one way to represent policy.
  - Frequently used mechanism for describing policy
- Columns are objects, subjects are rows.
- To determine if $S_i$ has right to access object $O_j$, find the appropriate entry.
- Succinct descriptor for O (ISI*IOL) entries
- Matrix for each right.

<table>
<thead>
<tr>
<th></th>
<th>$O_1$</th>
<th>$O_2$</th>
<th>$O_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>$S_2$</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>$S_3$</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
Access Matrix Protection System

• Protection State
  ‣ Current state of matrix

• Can modify the protection state
  ‣ Via protection state operations
  ‣ E.g., can create objects
  ‣ E.g., owner can add a subject, operation mapping for their objects

• Lampson’s “Protection” paper
  ‣ Can even delegate authority to perform protection state ops
Protection System

- Why is Protection State insufficient to enforce security?

- **Goal**: a protection state in which we can determine whether an unauthorized operation will ever be allowed (**Safety**)
Protection System Problems

- Protection system approach is inadequate for security
  - Suppose a process runs bad code
- Processes can change their own permissions
  - Processes may become untrusted, but can modify policy
- Processes, files, etc. are created and modified
  - Cannot predict in advance (safety problem)
- What do we need to achieve necessary controls?
Define and Enforce Goals

- Claim: *If we can define and enforce a security policy that ensures security goals, then we can prevent such attacks*

- How do we know what policy will be enforced?

- How do we know the enforcement mechanism will enforce policy as expected?
  - Look into this today

- How do we know the policy expresses effective goals?
  - Will look into this in depth later
Mandatory Protection System

- Is a *protection system* that can be modified only by *trusted administration* that consists of:
  - A *mandatory protection state* where the protection state is defined in terms of an immutable set of *labels* and the *operations that subject labels can perform on object labels*
  - A *labeling state* that assigns system subjects and objects to those labels in the mandatory protection state
  - A *transition state* that determines the legal ways that subjects and objects may be relabeled
- MPS is *immutable* to user-space process
Mandatory Protection System

This diagram illustrates the Mandatory Protection System with the following components:

- **Labeling State**
  - Process: newproc
  - Process: other

- **File: newfile**
- **File: acct**

- **Transition State**

- **Protection State**

The table shows the read and write permissions for different label states:

<table>
<thead>
<tr>
<th></th>
<th>secret</th>
<th>unclassified</th>
<th>trusted</th>
<th>untrusted</th>
</tr>
</thead>
<tbody>
<tr>
<td>secret</td>
<td>read</td>
<td>write</td>
<td>read</td>
<td></td>
</tr>
<tr>
<td>unclassified</td>
<td>read</td>
<td>write</td>
<td>read</td>
<td>read</td>
</tr>
<tr>
<td>trusted</td>
<td>write</td>
<td>read</td>
<td>write</td>
<td>write</td>
</tr>
<tr>
<td>untrusted</td>
<td>read</td>
<td>write</td>
<td>read</td>
<td>read</td>
</tr>
</tbody>
</table>

This diagram outlines the access control rules for different label states in the Mandatory Protection System.
Mandatory Protection State

- Immutable table of
  - Subject labels
  - Object labels
  - Operations authorized for former upon latter

- How can you use an MPS to control use of bad code?
  - E.g., Prevent modification of kernel memory?
Mandatory Protection State

- Immutable table of
  - Subject labels
  - Object labels
  - Operations authorized for former upon latter

- How can you use an MPS to control use of bad code?
  - E.g., Prevent modification of kernel memory?
  - Subject labels for all subjects running “bad code” are not allowed modify kernel memory
Mandatory Protection State

- Immutable table of
  - Subject labels
  - Object labels
  - Operations authorized for former upon latter

- How can you use an MPS to control use of bad code?
  - E.g., Prevent modification of kernel memory?
  - Subject labels for all subjects running “bad code” are not allowed modify kernel memory
    - Or that may run “bad code” (be compromised)
  - How do subjects (processes) get their labels?
Labeling State

• Immutable rules mapping
  ▸ Subjects to labels (in rows)
  ▸ Objects to labels (in columns)

• How can you use labeling state to control bad code?
  ▸ E.g., Prevent modification of kernel memory?
Labeling State

• Immutable rules mapping
  ‣ Subjects to labels (in rows)
  ‣ Objects to labels (in columns)

• How can you use labeling state to control bad code?
  ‣ E.g., Prevent modification of kernel memory?
  ‣ Assign all processes that may run bad code …
  ‣ With a label that cannot modify kernel memory
  ‣ What about objects created by these processes?
Protecting Good Code

• How can you use labeling state to prevent good code from going bad?
Protecting Good Code

• How can you use labeling state to prevent good code from going bad?
  ‣ E.g., Prevent dependence on untrusted input?
  ‣ Assign object labels to all objects that may be adversary-controlled
  ‣ Do not grant subject labels that should run good code access to those labels
  ‣ Verify that you are running good code (how?) and assign to one of these protected subject labels
  ‣ What integrity model does this approximate?
Protecting Good Code

- What if good code needs to access some adversary-controlled resources?
Mandatory Protection State

• What if good code needs to access some adversary-controlled resources?
  ‣ (1) if a process reads adversary-controlled object label, remove privileged permissions (e.g., to modify kernel memory)
  ‣ (2) if a process reads adversary-controlled object label, remove permission to write to any object that may be accessed by a subject whose label grants privileged permissions

• How do we achieve this change with the MPS?
Transition State

- Immutable rules mapping
  - Subject labels to conditions that change their subject labels
  - Object labels to conditions that change their object labels
- How can you use labeling state to control bad code?
  - E.g., Achieve (1) and (2)
Transition State

• Immutable rules mapping
  ‣ Subject labels to conditions that change their subject labels
  ‣ Object labels to conditions that change their object labels

• How can you use labeling state to control bad code?
  ‣ E.g., Achieve (1) and (2)
  ‣ Change subject label of subject accessing adversary-controlled resources to remove these permissions
  ‣ What integrity model does this approximate?
Transition State

- Is it possible to launch processes with more permissions than the invoker with MPS?
Managing MPS

- **Challenge**
  - Determining how to set and manage an MPS in a complex system involving several parties

- **Parties**
  - What does programmer know about deploying their program securely?
  - What does an OS distributor know about running a program in the context of their system?
  - What does an administrator know about programs and OS?
  - Users?
Managing MPS

- Current methods use dynamic analysis to setup MAC policies – run the program and collect the permissions used
  - Really a functional policy
Reference Monitor

• **Purpose:** Ensure enforcement of security goals
  ‣ Define goals in the **mandatory protection system**
  ‣ **Reference monitor** ensures enforcement

• *Every component that you depend upon to enforce your security goals must be a reference monitor*
Reference Monitor

- Components
  - Reference monitor interface (e.g., LSM)
  - Reference validation mechanism (e.g., SELinux)
  - Policy store (e.g., policy database)
Reference Monitor Guarantees

• **Complete Mediation**
  ‣ The reference validation mechanism must always be invoked

• **Tamperproof**
  ‣ The reference validation mechanism must be tamperproof

• **Verifiable**
  ‣ The reference validation mechanism must be subject to analysis and tests, the completeness of which must be assured
Complete Mediation

- Every security-sensitive operation must be mediated
  - What’s a “security-sensitive operation”?
Complete Mediation

• Every security-sensitive operation must be mediated
  ‣ What’s a “security-sensitive operation”?
  ‣ E.g., operation that may not be authorized for every subject

• How do we validate complete mediation?
Complete Mediation

• Every security-sensitive operation must be mediated
  ‣ What’s a “security-sensitive operation”?
  ‣ E.g., operation that may not be authorized for every subject

• How do we validate complete mediation?
  ‣ Every security-sensitive operation must be identified
  ‣ E.g., ensure every execution of that operation is checked

• **Mediation**: Does interface mediate?

• **Mediation**: On all resources?

• **Mediation**: Verifiably to enforce security goals?
Tamperproof

- Prevent modification by untrusted entities
  - Prevent modification of what?
Tamperproof

• Prevent modification by untrusted entities
  ‣ Prevent modification of what?
  ‣ Code and data that can affect reference monitor

• How to detect tamperproofing?
Tamperproof

- Prevent modification by untrusted entities
  - Prevent modification of what?
  - Code and data that can affect reference monitor
- How to detect tamperproofing?
  - Check for strong integrity guarantees (Biba)
  - Challenge: Often some untrusted operations are present
- **Tamperproof**: Is reference monitor protected?
- **Tamperproof**: Is system TCB protected?
Verification

- Determine correctness of code and policy
  - What defines correct code?
  - What defines a correct policy?

- Test and analyze reference validation mechanism
  - Does code/policy do its job correctly?
  - For all executions (completeness must be assured)

- **Verifiable**: Is TCB code base correct?

- **Verifiable**: Does the MPS enforce the system’s security goals?
Evaluation

- **Mediation**: Does interface mediate?
- **Mediation**: On all resources?
- **Mediation**: Verifably?
- **Tamperproof**: Is reference monitor protected?
- **Tamperproof**: Is system TCB protected?
- **Verifiable**: Is TCB code base correct?
- **Verifiable**: Does the MPS enforce the system’s security goals?
Take Away

• Mandatory Protection System
  ‣ Means to define security goals that applications cannot impact

• Reference Monitor Concept
  ‣ Requirements for a reference validation mechanism that can correctly enforce an MPS
  ‣ NOTE: This will be a major focus of this course

• Until we come up with coherent approach to validating MPS meets security goals and validating reference monitor guarantees, we will continue to have insecure systems
  ‣ That is the challenge of systems security research