Advanced Systems Security: Security Goals

Trent Jaeger
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University

February 4, 2010
A Secure System

- Consists of a mandatory protection system enforced by a reference validation mechanism that satisfies the reference monitor concept
- Can we choose a mandatory protection state to enforce?
- Why is one better than another?
Mandatory Protection States

- Is smaller state better?
  - Fewer labels? Fewer permissions?

- Are more isolated labels better?
  - Fewer permissions to other labels

- How are labels related?
  - Are some better than others?

- What about labeling state and transition state?
  - What dynamics make sense?
Choosing Security Goals

• How do we define security for our systems?
Access Matrix

<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></td>
</tr>
<tr>
<td>trusted</td>
<td>write</td>
<td></td>
<td>read write</td>
<td>write</td>
</tr>
<tr>
<td>untrusted</td>
<td>read</td>
<td>write</td>
<td>read</td>
<td>write</td>
</tr>
</tbody>
</table>
Access Matrix Goal

• Protection State represents policy, and the policy should enforce a goal

• Difficult to write a goal in terms of a low-level policy

• We want a goal to be easily understandable
  ‣ Policy need not be if it complies
Least Privilege

- The protection mechanism should force every process to operate with the minimum privileges needed to perform its task.

- Due to Saltzer and Schroeder (of Multics project)

- One of many “design principles” in their paper “The Protection of Information in Computer Systems” (1975)

- Others
  - Principle of Psychological Acceptability
  - Principle of Fail Safe Defaults
Least Privilege

• How to compute least privilege?
  ‣ Aim: Determines the permissions required for the program to run effectively

• Various systems provide support
  ‣ SELinux audit2allow: take denied permissions and add them to policy
  ‣ AppArmor Profile Wizard: Build an approximate profile statically and

• Run the program and see what permissions are used
Least Privilege

• Is a good goal because…

• Is a poor goal because…

• Can we use it to verify a policy is secure?
Least Privilege

• Is a good goal because…
  › Unnecessary permissions lead to problems (confused deputy)
  › Accounts for function

• Is a poor goal because…
  › Task permissions may conflict with security
  › How do we know when a permission is necessary, but makes the system insecure?

• Can we use it to verify a policy is secure?
  › No. *It defines a policy based on function, not security.*
Information Flow

- Another approach looks at the authorized flow of information among processes via objects
Idealized Security

- **Multilevel Security (MLS) for secrecy**
  - **Secrecy requirement**: Do not *leak* data to unauthorized principals
  - Only permit information to flow from less secret to more secret principals/objects
  - E.g., Can only read a file if your *clearance* dominates that of the file

- **Biba Integrity**
  - **Integrity requirement**: Do not *depend* on data from lower integrity principals
  - Only permit information to flow from high integrity to lower integrity
  - E.g., Can only read a file if your *integrity level* is dominated by the file’s
Information Flows

- **Secrecy** (MLS): If the OS permits a secret application/object to flow to a public application/object, then there may be a leak (e.g., Trojan horse)

- **Integrity** (Biba): If the OS permits a low integrity input to flow to a high integrity application/object, then there may be a dependency (e.g., buffer overflow)
Information Flow Goals

- Can be represented as a graph: \( G = (V, E) \)
  - A set of vertices (one for each label) and the directed edges between them (authorized operations)
    - Note that edge \((u, v)\) implies either that \(u\) writes to \(v\) or \(v\) reads from \(u\)

- A goal can be stated as reachability constraint in such a graph
  - No flow from \(x\) to \(y\) in \(G\)
  - Must flow from \(x\) to \(z\) in \(G\)

- A system is secure if all reachability constraints are satisfied
Practical vs. Ideal

- Do these idealized approaches based on information flow enable practical realization of OS enforcement?
- Secrecy is possible in some environments
  - Implemented in a paper world, previously
- Integrity has not been realized in practice
  - Many processes provide high integrity services to others
- Result: Depend on many applications to manage information flows
Example: logrotate

• *Logrotate* is a service that swaps logs

• It rotates logs through sequence
  ‣ Secrecy: Logs may span all security levels on system
  ‣ Thus, *logrotate* is trusted in SELinux

• It reads a configuration to tell it what to do
  ‣ Integrity: Logs must not leak into configuration files
  ‣ Configurations must not cause file leakage
SELinux/MLS Trusted Programs

• The OS trusts that privileged applications preserve system secrecy (30+ programs)

SELinux/MLS:

Policy management tools

- secadm, load_policy, settrans, setfiles, semanage, restorecon, newrole

Startup utilities

- bootloader, initrc, init, local_login

File tools

- dpkg_script, dpkg, rpm, mount, fsadm

Network utilities

- iptables, sshd, remote_login, NetworkManager

Auditing, logging services

- logrotate, klogd, auditd, auditctl

Hardware, device mgmt

- hald, dmidecode, udev, kudzu

Miscellaneous services

- passwd, tmpreaper, insmod, getty, consoletype, pam_console
Situation Is Much Worse

- **Clients**
  - Lots of client programs are entrusted with information with different secrecy/integrity requirements
  - Email, browser, IM, VOIP, …

- **Servers**
  - Historically, many servers have enforced security policies because they handle multiple clients
  - Web servers, databases, mail, repositories, …

- *Information flow alone is not enough to build a secure system!*
What Do We Do?

- **Unsatisfying Solution #1: Ignore Exceptions**
  - Processes are outside the model
  - E.g., so-called “Trusted Readers/Writers” in MLS
  - **Result:** Accept that some processes can violate policy, often blindly

- **Unsatisfying Solution #2: Dump Info Flow Entirely**
  - Change to a more general policy model
  - No meaningful security goal (is *least privilege* adequate?)
  - **Result:** Systems are complex -- 50K SELinux rules
Information Flow

- Is a good goal because…

- Is a poor goal because…

- Can we use it to verify a policy is correct?
Information Flow

• Is a good goal because…
  ‣ No false negatives – an attack requires an illegal information flow
  ‣ Can define data and functional security requirements

• Is a poor goal because…
  ‣ Function may conflict with security
  ‣ How do we know when a permission is illegal, but is necessary for functional requirements?

• Can we use it to verify a policy is correct?
  ‣ Yes. It defines a policy based on security. But what about exceptions?
Information Flow for Others

• Q: Can we use information flow goals for non-information flow models (e.g., access matrix)?
Noninterference

- Information Flow may not be enough
- An information flow graph may not include all possible edges (covert channels)
- May create an information flow through shared, physical resources
  - Use same disk device (storage channel)
  - Use same CPU (timing channel)
- A noninterference goal requires that the user $u$ outputs have no affect on user $u'$ behavior, if $u > u'$
  - User $u'$ is unaffected by $u$
Noninterference

• An operation authorized at runtime may enable a covert flow
  ‣ Allow u to write resource x
  ‣ Allow u’ to read resource y
  ‣ Where y is dependent on the operation of writing x
  ‣ Fill a disk; see a CPU delay; see a value of y that implies a value of x

• In general, \( \text{purge}(u, \text{hist.cmd}(u)) = \text{purge}(u’, \text{hist}) \) when \( \text{SC}(u) > \text{SC}(u’) \)
  ‣ u’ does not see output that is affected by any cmd(u)
Noninterference as a Goal

- Depends on traces of execution
  - So, such goals are difficult to express
  - No system noninterference model checking
    - Although we will discuss noninterference models for programs (decentralized label model)
    - Which are being applied to overt flows (decentralized information flow control)
- Can such goals be expressed and checked effectively?
Impact on Labeling and Transition

• How should security goals impact labeling and transitions?

• **Labeling** – assignment of objects to labels
  ‣ What are the requirements for various secrecy/integrity levels?

• **Transitions** – conditions that determine when a label will change
  ‣ What are the requirements for causing a label change?

• What experience do we have?
Labeling

- How do we choose the label of a subject or object when created?
  - Traditional: same label as creator

- Alternatives
  - Based on certification
    - E.g., code hashes for integrity verification
  - Based on location of created object
    - Store this with garbage, so make it garbage
  - Based on creator preference
    - Trust the creator to decide (within a range)
Transitions

• When do we change the label of a subject or object? And to what?
  ‣ Traditional: don’t change

• Alternatives
  ‣ Based on operations performed by subject or on object
    • Low-water mark integrity; High-water mark secrecy
    • There are access control models based on these
  ‣ Based on operations of others
    • E.g., When another process receives untrusted input, downgrade the whole system
Take Away

• To build a secure system, we need to define a mandatory protection system
  ▸ But, what should guide the design?

• Security goals
  ▸ Usually serves as a rough guide

• Types of security goals – biased toward security or function
  ▸ Functional: least privilege; Security: information flow

• Need to develop approaches to design goals for entire mandatory protection system – from function and security