CSE543 - Introduction to Computer and Network Security

Module: Access Control

Professor Trent Jaeger
Policy

• A policy specifies the rules of security
  ‣ Some statement of secure procedure or configuration that parameterizes the operation of a system
  ‣ Example: Airport Policy
    • Take off your shoes
    • No bottles that could contain $> 3$ ozs
    • Empty bottles are OK?
    • You need to put your things through X-ray machine
    • Laptops by themselves, coat off
    • Go through the metal detector

• **Goal**: prevent on-airplane (metal) weapon, flammable liquid, dangerous objects … (successful?)
Computer Security Policy Goals

- **Secrecy**
  - Don’t allow reading by unauthorized subjects
  - Control where data can be written by authorized subjects
    - Why is this important?

- **Integrity**
  - Don’t permit dependence on lower integrity data/code
    - Why is this important?
  - What is “dependence”?

- **Availability**
  - The necessary function must run
  - Doesn’t this conflict with above?
… when policy goes wrong

- Driving license test: take until you pass
  - Mrs. Miriam Hargrave of Yorkshire, UK failed her driving test 39 times between 1962 and 1970!!!
  - … she had 212 driving lessons ….
  - She finally got it on the 40th try.
  - Some years later, she was quoted as saying, “sometimes I still have trouble turning right”

“A policy is a set of acceptable behaviors.”

- F. Schneider
Access Control/Authorization

- Access control/authorization determines what rights (operations) a particular subject has for a set of objects.
- It answers the question:
  - E.g., do you have the right to read /etc/passwd?
  - Does Alice have the right to view the CSE website?
  - Do students have the right to share project data?
  - Does Dr. Jaeger have the right to change your grades?

- An Access Control Policy answers these questions.
Simplified Access Control

• **Subjects** are the active entities that do things
  ‣ E.g., you, Alice, students, Prof. Jaeger

• **Objects** are passive things that things are done to
  ‣ E.g., /etc/passwd, CSE website, project data, grades

• **Rights (operations)** are actions that are taken
  ‣ E.g., read, view, share, change
Trusted Computing Base (TCB)

- The trusted computing base is the infrastructure that you assume will behave correctly
  - Hardware (keyboard, monitor, …)
  - Operating Systems
  - Implementations
  - Local networks
  - Administrators
  - Other users on the same system

- Axiom: the larger the TCB, the more assumptions you must make (and hence, the more opportunity to have your assumptions violated).
Protection Domains

- The *protection domain* restricts access of processes to our computing system’s resources.
- How is this done today?
  - Memory protection
  - E.g., UNIX protected memory, file-system permissions (rwx...)

*Policy is defined with respect to the protection domain it governs.*
Protection Domains

Protection domain

- The protection domain restricts access of processes to our computing system’s resources

- How is this done today?
  - Memory protection
  - E.g., UNIX protected memory, file-system permissions (rwx...)

Policy is defined with respect to the protection domain it governs.
Access Policy Enforcement

• A protection state defines what each subject can do
  ‣ E.g., in mode bits, ACLs, ... --- the policy

• A reference monitor enforces the protection state
  ‣ A service that responds to the query...

• A correct reference monitor implementation meets the following guarantees
  ‣ Tamperproof
  ‣ Complete Mediation
  ‣ Simple enough to verify

• A protection system consists of a protection state, operations to modify that state, and a reference monitor to enforce that state
The Access Matrix

- An access matrix is one way to represent policy.
  - Frequently used mechanism for describing policy (protection state)
- Columns are objects, subjects are rows.
  - To determine if $S_i$ has right to access object $O_j$, find the appropriate entry.
  - There is a matrix for each right.

- The access matrix is a succinct descriptor for $O(|S|*|O|)$ rules

\[
\begin{array}{c|ccc}
  & O_1 & O_2 & O_3 \\
\hline
S_1 & Y & Y & N \\
S_2 & N & Y & N \\
S_3 & N & Y & Y \\
\end{array}
\]
### Access Matrix

- Suppose the private key file for J is object $O_1$
  - Only J can read
- Suppose the public key file for J is object $O_2$
  - All can read, only J can modify
- Suppose all can read and write from object $O_3$
- What’s the access matrix?

<table>
<thead>
<tr>
<th></th>
<th>$O_1$</th>
<th>$O_2$</th>
<th>$O_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>$S_2$</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>$S_3$</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
Secrecy

- Does the following protection state ensure the secrecy of J’s private key in $O_1$?

<table>
<thead>
<tr>
<th></th>
<th>$O_1$</th>
<th>$O_2$</th>
<th>$O_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>J</strong></td>
<td>R</td>
<td>RW</td>
<td>RW</td>
</tr>
<tr>
<td>$S_2$</td>
<td>-</td>
<td>R</td>
<td>RW</td>
</tr>
<tr>
<td>$S_3$</td>
<td>-</td>
<td>R</td>
<td>RW</td>
</tr>
</tbody>
</table>
Integrity

• Does the following access matrix protect the integrity of J’s public key file \(O_2\)?

<table>
<thead>
<tr>
<th></th>
<th>(O_1)</th>
<th>(O_2)</th>
<th>(O_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(J)</td>
<td>R</td>
<td>RW</td>
<td>RW</td>
</tr>
<tr>
<td>(S_2)</td>
<td>-</td>
<td>R</td>
<td>RW</td>
</tr>
<tr>
<td>(S_3)</td>
<td>-</td>
<td>R</td>
<td>RW</td>
</tr>
</tbody>
</table>
Trusted Processes

- Does it matter if we do not trust some of J’s processes?

- **Trojan Horse**: Attacker controlled code run by J can violate secrecy

- **Confused Deputy**: Attacker may trick untrusted code to violate integrity
Protection vs Security

- Protection
  - Security goals met under *trusted* processes
  - Protects against an error by a non-malicious entity

- Security
  - Security goals met under *potentially malicious* processes
  - Protects against any malicious entity

- Hence, For J:
  - Non-malicious process shouldn’t leak the private key by writing it to $O_3$
  - A potentially malicious process may contain a Trojan horse that can write the private key to $O_3$
A system should only provide those rights needed to perform the processes function and no more.

- **Implication 1**: you want to reduce the protection domain to the smallest possible set of objects
- **Implication 2**: you want to assign the minimal set of rights to each object
- **Caveat**: of course, you need to provide enough rights and a large enough protection domain to get the job done.
Least Privilege

• Limit permissions to those required and no more
• Consider three processes for user J
  ‣ Restrict privilege of the process J₁ to prevent leaks

<table>
<thead>
<tr>
<th></th>
<th>O₁</th>
<th>O₂</th>
<th>O₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>J₁</td>
<td>R</td>
<td>RW</td>
<td>-</td>
</tr>
<tr>
<td>J₂</td>
<td>-</td>
<td>R</td>
<td>-</td>
</tr>
<tr>
<td>J₃</td>
<td>-</td>
<td>R</td>
<td>RW</td>
</tr>
</tbody>
</table>
Conflicting Goals

- Challenges of building a secure system
  - What are the users’ goals?
  - What do application developers want?
  - What about the data owners (corporations/governments)?
  - What is the purpose of system administrators?
  - What about the requirements of operating system designers?

- Need a satisfying balance among these goals?
Access Control Administration

There are two central ways to specify a policy

1. **Discretionary** - object “owners” define policy
   - Users have discretion over who has access to what objects and when (trusted users)
   - Canonical example, the UNIX filesystem
     - RWX assigned by file owners

2. **Mandatory** - Environment enforces static policy
   - Access control policy defined by environment, user has no control over access control (untrusted users)
   - Canonical example, process labeling
     - System assigns labels for processes, objects, and a dominance calculus is used to evaluate rights
**DAC vs. MAC**

- **Discretionary Access Control**
  - User defines the access policy
  - Can pass rights onto other subjects (called *delegation*).
  - Their programs can pass their rights
    - Consider a Trojan horse.

- **Mandatory Access Control**
  - System defines access policy
  - Subjects cannot pass rights
  - Subjects’ programs cannot pass rights
    - Consider a Trojan horse here.
Administrative Operations

• An access matrix defines a protection state
• A protection system also includes a set of operations for modifying that state
• Examples
  ‣ Add right (UNIX): If the user is the owner of the object, then the user can add an operation to set of operations of another user
  ‣ Add right: If domain has the copy flag set for that right in its access matrix row, then it can add that right to any other domain’s access row
DAC vs. MAC in Access Matrix

- **Subjects:**
  - DAC: users
  - MAC: labels

- **Objects:**
  - DAC: files, sockets, etc.
  - MAC: labels

- **Operations:**
  - Same

- **Administration:**
  - DAC: owner, copy flag, ...
  - MAC: external

- **MAC:** largely static matrix; DAC: all can change

```
+-----+-----+-----+
|     | O1   | O2   | O3   |
|-----+-----+-----+-----|
| S1  | Y    | Y    | N    |
| S2  | N    | Y    | N    |
| S3  | N    | Y    | Y    |
```
Safety Problem

- For a protection system
  ‣ (ref mon, protection state, and administrative operations)
- Prove that any future state will not result in the leakage of an access right to an unauthorized user
  ‣ Q: Why is this important?
- For most discretionary access control models,
  ‣ Safety is *undecideable*
- Means that we need another way to prove safety
  ‣ Restrict the model (no one uses)
  ‣ Test incrementally (constraints)
- How does the safety problem affect MAC models?
Access Control Models

• What language should I use to express policy?
  ‣ Access Control Model

• Oodles of these
  ‣ Some specialize in secrecy
    • Bell-LaPadula
  ‣ Some specialize in integrity
    • Clark-Wilson
  ‣ Some focus on jobs
    • RBAC
  ‣ Some specialize in least privilege
    • SELinux Type Enforcement

• Q: Why are there so many different models?
Groups

• Groups are collections of identities who are assigned rights as a collective
• Important in that it allows permissions to be assigned in aggregates of users …

• This is really about “membership”
  ‣ Standard DAC
  ‣ Permissions are transient
Job Functions

• In an enterprise, we don’t really do anything as ourselves, we do things as some job function
  ‣ E.g., student, professor, doctor

• One could manage this as groups, right?
  ‣ We are assigned to groups all the time, and given similar rights as them, i.e., mailing lists
Roles

- A role is a collection of privileges/permissions associated with some function or affiliation
- NIST studied the way permissions are assigned and used in the real world, and this is it ...

- Important: the permissions are static, the user-role membership is transient
- This is not standard DAC
Role Based Access Control

- Role based access control is a class of access control not direct MAC and DAC, but may one or either of these.
- A lot of literature deals with RBAC models.
- Most formulations are of the type:
  - $U$: users -- these are the subjects in the system
  - $R$: roles -- these are the different roles users may assume
  - $P$: permissions --- these are the rights which can be assumed
- There is a many-to-many relation between:
  - Users and roles
  - Roles and permissions
- Relations define the role-based access control policy
RBAC Sessions

• During a session, a user assumes a subset available roles
  ‣ Known as activating a set of roles
  ‣ The user rights are the union of the rights of the activated roles
  ‣ Note: the session terminates at the user’s discretion

• Q: Why not just activate all the roles?
Constraints

• You want to constrain evolution of protection states
  ‣ Constraints are explicit ways of doing just this
  ‣ Constraints available (in RBAC)
    • role assumption
    • perm-role assignment
    • user-role assignment

• Examples in RBAC:
  ‣ Required inclusion: You must be acting as an employee of Pennsylvania State University to be a professor
    • You must assume a (parent) role to assume another (child) role
  ‣ Mutual exclusion: can not be both CFO and auditor for the same company (unless you work for Enron)
  ‣ Cardinality constraint: only one (or n) of a particular role
Constraint Example

- **Mutual Exclusion**: No entity can activate student and faculty roles at the same time?
  - Give yourself credits, etc.
  - Or, in this case buy faculty tickets at student prices?
Multilevel Security

- A multi-level security system tags all object and subject with security tags classifying them in terms of sensitivity/access level.
  - We formulate an access control policy based on these levels
  - We can also add other dimensions, called categories which horizontally partition the rights space (in a way similar to that as was done by roles)
US DoD Policy

- Used by the US military (and many others), uses MLS to define policy

- Levels:

  UNCLASSIFIED < CONFIDENTIAL < SECRET < TOP SECRET

- Categories (actually unbounded set)

  NUC(lear), INTEL(igence), CRYPTO(graphy)

- Note that these levels are used for physical documents in the governments as well.
Assigning Security Levels

• All subjects are assigned clearance levels and compartments
  ‣ Alice: (SECRET, {CRYPTO, NUC})
  ‣ Bob: (CONFIDENTIAL, {INTEL})
  ‣ Charlie: (TOP SECRET, {CRYPTO, NUC, INTEL})

• All objects are assigned an access class
  ‣ DocA: (CONFIDENTIAL, {INTEL})
  ‣ DocB: (SECRET, {CRYPTO})
  ‣ DocC: (UNCLASSIFIED, {NUC})
Evaluating Policy

• Access is allowed if

subject clearance level $\geq$ object sensitivity level and
subject categories $\supseteq$ object categories (read down)

• Q: What would write-up be?
Bell-La Padula Model

- A Confidentiality MLS policy that enforces:
  - *Simple Security Policy*: a subject at specific classification level cannot read data with a higher classification level. This is shorthand for “*no read up*”.
  - * (star) Property*: also known as the confinement property, states that subject at a specific classification cannot write data to a lower classification level. This is shorthand for “*no write down*”.

```
(Top Secret, {nuclear,crypto})
(Top Secret, {nuclear})
(Secret, {nuclear,crypto})
(Secret, {nuclear})
(Top Secret, {crypto})
(Secret, {crypto})
(Secret, {})
(Top Secret, {})
```
How about integrity?

• MLS as presented before talks about who can “read” a document (confidentiality)

• Integrity considers who can “write” to a document
  ‣ Thus, who can affect the integrity (content) of a document
  ‣ Example: You may not care who can read DNS records, but you better care who writes to them!

• Biba defined a dual of secrecy for integrity
  ‣ Lattice policy with, “no read down, no write up”
    • Users can only *create* content at or *below* their own integrity level (a monk may write a prayer book that can be read by commoners, but not one to be read by a high priest).
    • Users can only *view* content at or *above* their own integrity level (a monk may read a book written by the high priest, but may not read a pamphlet written by a lowly commoner).
Integrity, Sewage, and Wine

- Mix a gallon of sewage and one drop of wine gives you?
- Mix a gallon of wine and one drop of sewage gives you?

*Integrity is really a contaminant problem:* you want to make sure your data is not contaminated with data of lower integrity.
Biba (example)

• Which users can modify what documents?
  ‣ Remember “no read down, no write up”

Bob: (CONF., {INTEL})
Charlie: (TS, {CRYPTO, NUC, INTEL})
Alice: (SEC., {CRYPTO, NUC})

DocA: (CONFIDENTIAL, {INTEL})
DocB: (SECRET, {CRYPTO})
DocC: (UNCLASSIFIED, {NUC})
- **Low-Water Mark integrity**
  - Change integrity level based on actual dependencies

- Subject is initially at the highest integrity
  - But integrity level can change based on objects accessed

- Ultimately, subject has integrity of lowest object read
Clark-Wilson Integrity

- Map Integrity in Business (e.g., accounting) to Computing
- High Integrity Data (objects)
  - "Constrained Data Items" (CDIs)
- High Integrity Processes (programs)
  - "Transformation Procedures" (TPs)
- Check Integrity of Data Initially (verification)
  - "Integrity Verification Procedures" (IVPs)
- Premise
  - If the IVPs verify initial integrity
  - and high integrity data is only modified by TPs
  - Then, the integrity of computation is preserved
Clark-Wilson Permissions

Diagram showing the relationship between users and CDIs.
CW Permissions (cont.)

Diagram showing the relationship between User, TP, and CDI roles, with arrows indicating permissions or access.
CW Permissions (cont.)

• A user can access an CDI using TP iff
  1. The user has been granted CDI access
  2. The TP has been granted CDI access
  3. The user has been granted access to the TP
Clark-Wilson Issues

• Assure Function
  ‣ Certify IVPs, TPs to be ‘valid’ (i.e., correct) (C1,C2)
  ‣ Is there a general way of defining correctness?

• Handle Low Integrity Data
  ‣ A TP must upgrade or discard any UDI (low integrity data) it receives (C5)

Reality: this is a nice model, but too heavyweight in general for most applications. CW-lite (Jaeger) is an alternative that is tractable to implement.