Operating System Security

Chester Rebeiro
IIT Madras
Security Goals

Protecting data and resources has three requirements

- **Secrecy (confidentiality)**
  - Unauthorized disclosure
  - Limits the objects (files/sockets) that a process can read

- **Integrity**
  - Unauthorized modification
  - Limits the objects that a process can write
    (objects may contain information that other processes depend on)

- **Availability**
  - Limits the system resources that processes (or users) may consume
  - Therefore preventing denial of service attacks
  - Achieved by OS resource management techniques like fair scheduling
Confidentiality & Integrity

Achieved by Access Control

• Every access to an object in the system should be controlled

• All and only authorized accesses can take place

Access ? Specifying an operation on the object like read, write, execute, create, delete
Access Control Systems

• Development of an access control system has three components
  – **Security Policy**: high level rules that define access control
  – **Security Model**: a formal representation of the access control security policy and its working.
    (this allows a mathematical representation of a policy; there by aid in proving that the model is secure)
  – **Security Mechanism**: low level (sw / hw) functional implementations of policy and model
Security Policy

• A scheme for specifying and enforcing security policies in a system

• Driven by
  – Understanding of threat and system design

• Often take the form of a set of statements
  – Succinct statements
  – Goals are agreed upon either by
    • The entire community
    • Top management
    • Or is the basis of a formal mathematical analysis
A bad security policy model of a company

Who determines need to know?

Who enforces this?

How are breaches detected?
Who’s duty is it to report them?

Approval of policy should not be part of the policy document itself.

Megacorp Inc security policy
1. This policy is approved by Management.
2. All staff shall obey this security policy.
3. Data shall be available only to those with a ‘need-to-know’.
4. All breaches of this policy shall be reported at once to Security.
Security Model

- Why have it at all?
  - It is a mathematical representation of the policy.
  - By proving the model is secure and that the mechanism correctly implements the model, we can argue that the system is indeed secure (w.r.t. the security policy)
Security Mechanism

• Implementing a correct mechanism is non trivial
• Could contain bugs in implementation which would break the security
• The implementation of the security policy must work as a ‘trusted base’ (reference monitor)
• Properties of the implementation
  o Tamper proof
  o Non-bypassable (all accesses should be evaluated by the mechanism)
  o Security kernel – must be confined to a limited part of the system (scattering security functions all over the system implies that all code must be verified)
  o Small – so as to achieve rigorous verification
Access Control Techniques

- DAC – Discretionary
- MAC – Mandatory
- RBAC -- Role-based
Discretionary Access Control

- Discretionary (DAC)
  - Access based on
    - Identity of requestor
    - Access rules state what requestors are (or are not) allowed to do
  - Privileges granted or revoked by an administrator
  - Users can pass on their privileges to other users
  - Example. Access Matrix Model
Access Matrix Model

- By Butler Lampson, 1971
- Subjects: active elements requesting information
- Objects: passive elements storing information
  - Subjects can also be objects
- Other actions: ownership (property of objects by a subject), control (father-children relationships between processes)

Butler Lampson, “Protection”, 1971
A formal representation of Access Matrix Model

- Define an access matrix: $A[X_{si}, X_{oj}]$
- Protection System consists of
  - **Generic rights**: $R = \{r_1, r_2, \cdots, r_k\}$ thus $A[X_{si}, X_{oj}] \subseteq R$
  - **Primitive Operations**: $O = \{op_1, op_2, \cdots, op_n\}$

<table>
<thead>
<tr>
<th>subjects</th>
<th>File 1</th>
<th>File 2</th>
<th>File 3</th>
<th>Program 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ann</td>
<td>own read write</td>
<td>read</td>
<td>execute</td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td>read</td>
<td>read write</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Carl</td>
<td>read</td>
<td>execute read</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Generic rights**: enter $r$ into $A[X_{si}, X_{oj}]$, delete $r$ from $A[X_{si}, X_{oj}]$, create subject $X_s$, create object $X_o$, destroy subject $X_s$, destroy object $X_o$
A formal representation of Access Matrix Model

• Commands

\[
\text{command } \alpha(X_1, X_2, \ldots, X_n)
\]
\[
\text{if } r_1 \text{ in } A[X_{s_1}, X_{o_1}] \text{ and } \\
\quad r_2 \text{ in } A[X_{s_2}, X_{o_2}] \text{ and } \\
\quad r_3 \text{ in } A[X_{s_3}, X_{o_3}] \text{ and } \\
\quad \vdots \quad \vdots \quad \vdots \\
\quad r_k \text{ in } A[X_{s_k}, X_{o_k}] \\
\text{then } \quad op_1 \\
\quad op_2 \\
\quad op_3 \\
\quad \vdots \\
\text{end}
\]

access matrix \[ A[X_{s_i}, X_{o_j}] \]

Generic rights \[ R = \{r_1, r_2, \ldots, r_k\} \]

Primitive operations \[ O = \{op_1, op_2, \ldots, op_n\} \]

- enter \( r \) into \( A[X_{s_i}, X_{o_j}] \)
- delete \( r \) from \( A[X_{s_i}, X_{o_j}] \)
- create subject \( X_s \)
- create object \( X_o \)
- destroy subject \( X_s \)
- destroy object \( X_o \)
Example Commands

command $\alpha(X_1, X_2, \ldots, X_n)$

\[
\begin{align*}
\text{if } r_1 \text{ in } A[X_{s1}, X_{o1}] \text{ and } \\
r_2 \text{ in } A[X_{s2}, X_{o2}] \text{ and } \\
r_3 \text{ in } A[X_{s3}, X_{o3}] \text{ and } \\
\vdots & \quad \vdots \\
\text{then } & \quad op_1 \nonumber \\
op_2 \nonumber \\
op_3 \nonumber \\
\vdots & \quad \vdots \\
\text{end} \nonumber
\end{align*}
\]

Create an object

```
command CREATE(process, file)  
create object file  
enter own into (process, file)  
end
```

Confer ‘r’ right to a friend for the object

```
command CONFER, (owner, friend, file)  
if own in (owner, file)  
then enter r into (friend, file)  
end
```

Owner can revoke Right from an ‘ex’friend

```
command REMOVE, (owner, exfriend, file)  
if own in (owner, file) and  
r in (exfriend, file)  
then delete r from (exfriend, file)  
end
```
States of Access Matrix

• A protection system is a state transition system

• Leaky State:
  – A state (access matrix) is said to leak a right ‘r’ if there exists a command that adds right ‘r’ into an entry in the access matrix that did not previously contain ‘r’
  – Leaks may not be always bad.
Is my system safe?

• Safety
  – *Definition 1*: System is safe if access to an object without owner’s concurrence is impossible
  – *Definition 2*: A user should be able to tell if giving away a right would lead to further leakage of that right.
Safety in the formal model

• Suppose a subject $s$ plans to give subjects $s'$ right $r$ to object $o$.
  – with $r$ entered into $A[s', o]$, is such that $r$ could subsequently be entered somewhere new.
  – If this is possible, then the system is unsafe
Unsafe State (Example)

• Consider a protection system with two commands

```plaintext
command CONFERENCEexecute(S, S', O)
  if o in A[S, O] then
    enter x in A[S', 0]
  end

command MODIFY_RIGHT(S, O)
  if x in A[S, O] then
    enter w in A[S, O]
  end
```

• Scenario: Bob creates an application (object). He wants it to be executed by all others but not modified by them.

• The system is unsafe due to the presence of MODIFY_RIGHT in the protection system.
  – Alice could invoke MODIFY_RIGHT to get modification rights for the application.
Safety Theorem

• Given an initial state of the matrix (say \( A_0 \)) and a right ‘\( r \)’, we say that the state \( A_0 \) is unsafe if there exists a state \( A_i \) such that,
  1. \( A_i \) is reachable from \( A_0 \)
     • There are a sequence of transitions (commands) that would take the state from \( A_0 \) to \( A_i \)
  2. \( A_i \) leaks ‘\( r \)’

Determining if a system is safe is undecidable
Access Matrix Model Implementation (Authorization Table)

• Matrix not efficient
  – Too large and too sparse
• Authorization Table
  – Used in databases
  – Needs to search entire table in order to identify access permission

<table>
<thead>
<tr>
<th>USER</th>
<th>ACCESS MODE</th>
<th>OBJECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ann</td>
<td>own</td>
<td>File 1</td>
</tr>
<tr>
<td>Ann</td>
<td>read</td>
<td>File 1</td>
</tr>
<tr>
<td>Ann</td>
<td>write</td>
<td>File 1</td>
</tr>
<tr>
<td>Ann</td>
<td>read</td>
<td>File 2</td>
</tr>
<tr>
<td>Ann</td>
<td>write</td>
<td>File 2</td>
</tr>
<tr>
<td>Ann</td>
<td>execute</td>
<td>Program 1</td>
</tr>
<tr>
<td>Bob</td>
<td>read</td>
<td>File 1</td>
</tr>
<tr>
<td>Bob</td>
<td>read</td>
<td>File 3</td>
</tr>
<tr>
<td>Bob</td>
<td>write</td>
<td>File 3</td>
</tr>
<tr>
<td>Carl</td>
<td>read</td>
<td>File 2</td>
</tr>
<tr>
<td>Carl</td>
<td>execute</td>
<td>Program 1</td>
</tr>
<tr>
<td>Carl</td>
<td>read</td>
<td>Program 1</td>
</tr>
</tbody>
</table>
Access Matrix Model Implementation (Capabilities)

- **Capabilities**
  - Each user associated with a capability list, indicating, for each object the permitted operations.
  - Advantage in distributed systems, since it prevents repeated authentication of a subject.
  - Vulnerable to forgery: can be copied and misused by an attacker.
Access Matrix Model Implementation (ACL)

- Access Control Lists
  - Each object is associated with a list indicating the operations that each subject can perform on it
  - Easy to represent by small bit-vectors
ACL Implementation in Unix

• Users belong to exactly one group
• Each file has an owner
• Authorization for each file can be specified
  – For file’s owner (r,w,x $\rightarrow$ 3 bits)
  – For the group (r,w,x $\rightarrow$ 3 bits)
  – For the rest of the world (r,w, x $\rightarrow$3 bits)
Vulnerabilities in Discretionary Policies

• Subjected to Trojan Horse attacks
  – A Trojan horse can inherit all the user’s privileges
  – Why?
    • A trojan horse process started by a user sends requests to OS on the user’s behalf
Drawback of Discretionary Policies

• It is not concerned with information flow
  – Anyone with access can propagate information

• Information flow policies
  – Restrict how information flows between subjects and objects
Information Flow Policies

- Every object in the system assigned to a security class (SC)

Security classes (SC)

- Information flow triple:
  \( \langle SC, \rightarrow, \oplus \rangle \)
  \( \rightarrow \) is the can flow relation
  - \( B \rightarrow A \) : Information from \( B \) can flow to \( A \)
  - \( C \rightarrow B \rightarrow A \) : Information flow
  - \( C \leq B \leq A \) : Dominance relation
  \( \oplus \) is the join relation
  - defines how to label information obtained by combining information from two classes
  - \( \oplus : SC \times SC \rightarrow SC \).

\( SC, \rightarrow, \oplus \) are fixed and do not change with time.
The \( SC \) of an object may vary with time.

Examples

• Trivial case (also the most secure)
  – No information flow between classes
    
    \[ SC = \{ A_1 (low), A_2, \cdots, A_n (high) \} \]
    \[ A_i \rightarrow A_i \text{ (for } i = 1 \cdots n) \]
    \[ A_i \oplus A_i = A_i \]

• High to Low flows only
  
  \[ SC = \{ A_1 (low), A_2, \cdots, A_n (high) \} \]
  \[ A_j \rightarrow A_i \text{ only if } j \leq i \text{ (for } i, j = 1 \cdots n) \]
  \[ A_i \oplus A_j = A_i \]
Examples

• A company has the following security policy
  – A document made by a manager can be read by other managers but no workers
  – A document made by a worker can be read by other workers but no managers
  – Public documents can be read by both Managers and Workers

• What are the security classes?
• What is the flow operator?
• What is the join operator?
Examples

- A company has the following security policy
  - A document made by a manager can be read by other managers but no workers
  - A document made by a worker can be read by other workers but no managers
  - Public documents can be read by both Managers and Workers

\[
\begin{align*}
  \text{SC} &= \{P\text{ (low)}, W, M\text{ (high)}\} \\
  P &\rightarrow M, P \rightarrow W, W \rightarrow W, M \rightarrow M \\
  P \oplus M &\rightarrow M, P \oplus W \rightarrow W, M \oplus M \rightarrow M, W \oplus W \rightarrow W
\end{align*}
\]
Mandatory Access Control

• Access based on regulations set by a central authority
• Most common form is multilevel security (MLS) policy
  – Access Class
    • Objects need a **classification level**
    • Subjects needed a **clearance level**
  – A subject with $X$ clearance can access all objects in $X$ and below $X$ but not vice-versa
  – Information only flows upwards and cannot flow downwards
Bell-LaPadula Model

- Developed in 1974
- Formal model for access control
- Four access modes:
  - read, write, append, execute
- Two properties (MAC rules)
  - No read up (simple security property (ss-property))
  - No write down (*-property)

D. E. Bell and L. J. LaPadula, Secure Computer System: Unified
No read up

- Can only read confidential and unclassified files
No Write Down

- Cannot write into an unclassified object
Why No Write Down?

- A process inflected with a trojan, could read confidential data and write it down to unclassified.
- We trust users but not subjects (like programs and processes).
Discretionary Access Control

- An individual may grant access to a document he/she owns to another individual.
- However the MAC rules must be met.

MAC rules over rides any discretionary access control. A user cannot give away data to unauthorized persons.
Limitations of BLP

• Write up is possible with BLP
• Does not address Integrity Issues

User with clearance can modify a secret document
BLP only deals with confidentiality. Does not take care of integrity.
Biba Model

- Bell-LaPadula upside down
- **Ignores confidentiality and only deals with integrity**
- **Goals of integrity**
  - Prevent unauthorized users from making modifications in a document
  - Prevent authorized users from making improper modifications in a document
- **Incorporated in Microsoft Windows Vista**
BIBA Properties

No read down: Simple Integrity Theorem
No write up: * Integrity Theorem
Why no Read Down?

• A higher integrity object may be modified based on a lower integrity document
Example

Read Up
• A document from the general should be read by all

No Read Down
• A private’s document should not affect the General’s decisions
Secure Operating Systems

• A secure OS has 3 requirements
  – Complete mediation
    • Access enforcement mechanisms of OS should mediate all security-sensitive operations.
  – Tamperproof
    • Access enforcement mechanisms of OS should not be modifiable by an untrusted process
  – Verifiable
    • The access enforcement mechanisms of OS must be small enough to be completely and thoroughly tested.

How to precisely achieve these requirements!!
Complete Mediation

• Where to mediate is crucial!
• Trivial approach: mediate all system calls as these are entry points
  – Insufficient :
    • example, for the open system call,
      
      ```python
      open('/home/user/Desktop/file.txt', 'w')
      ```
      
      does the user have write permission for each directory object along the path
    
    permission to access file.txt may change from the start of the open system call to the actual open operation.
    
    • Reference monitors have to be embedded in the OS. Each system call needs to be considered independently.
Linux Reference Monitor (LSM)

- **LSM**: Linux Security Module is the reference module for Linux.
- Every system call will have a hook that invokes the reference monitor.
- LSM does not authorize open system call, but each individual directory, link, and file open after the system object reference has been retrieved.

Placement of LSM hook is important?
Tamperproof

• Reference monitor should not be modifiable outside the trusted computing base.
  – This must be verified.
  – Verification tool itself must be tamperproof
Threats

• Control flow hacking
  – Example: Buffer overflows
• Covert Channels