Goals for Today

- **Learning Objective:**
  - Compare cryptography and access control as means of providing security assurances
  - Understand fundamentals of access control within the operating system

- **Announcements, etc:**
  - MP3 is now available for download on Compass!
    - **DUE APRIL 15th (5 days from now)**
  - MP2.5 enrollment now closed!
  - MP4 Release Date — **April 17**
  - MP4 Q&A — **April 17**
  - Final Exam — **MAY 7TH @ 7PM, SIEBEL 1404**

**Reminder:** Please put away devices at the start of class
CS 423
Operating System Design: Access Control in Operating Systems

Professor Adam Bates
Security Goals

- **Confidentiality:**
  - Prevent data exposure

- **Integrity:**
  - Prevent data tampering

- **Authenticity:**
  - Verify identity of data source

- **Availability:**
  - Preventing denial of service
  - i.e., “Unplug computer” != security
Two competing primitives for achieving security

Both can be used to achieve assurances of data confidentiality, integrity, authenticity

Crypto: “Encrypt the data for security!”
  - Key is authorization mechanism to assure confidentiality, integrity, authenticity properties.

Access Control: “Label the data for security!”
  - Software includes authorization mechanism to assure confidentiality, integrity, authenticity properties according to policy.
• Two competing primitives for achieving security
• Both can be used to achieve assurances of data confidentiality, integrity, authenticity
• Crypto- and AC-based architectures are often very similar:

```
Client A  -->  Kerberos Server  -->  Remote Resource B
```

Read Request, A<->B Key
Resource B

“I’d like read access to Resource B”
Authorized (Return A<->B key)
• Two competing primitives for achieving security
• Both can be used to achieve assurances of data confidentiality, integrity, authenticity
• Crypto- and AC- based architectures are often very similar:

```
Client A

"I'd like read access to Resource B"
Authorized (Return capability)

CAPABILITY SYSTEM

Remote Resource B

Access Request, capability for B
Resource B
```
- Two competing primitives for achieving security
- Both can be used to achieve assurances of data confidentiality, integrity, authenticity
- Crypto- and AC- based architectures are often very similar:
<table>
<thead>
<tr>
<th>Cryptography</th>
<th>Access Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easily supports multiple administrative domains</td>
<td>Struggles to support multiple administrative domains</td>
</tr>
<tr>
<td>Easily supports delegation (i.e., share key)</td>
<td>Easily supports centralized global security policy</td>
</tr>
<tr>
<td>Computationally expensive</td>
<td>Computationally cheap</td>
</tr>
<tr>
<td>Requires key management and distribution mechanisms</td>
<td>Requires policy management mechanisms</td>
</tr>
<tr>
<td>Requires key security</td>
<td>Requires reference monitor</td>
</tr>
</tbody>
</table>
## Crypto v. Access Control

**Which primitive should we use in operating systems?**

<table>
<thead>
<tr>
<th>Cryptography</th>
<th>Access Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easily supports multiple administrative domains</td>
<td>Struggles to support multiple administrative domains</td>
</tr>
<tr>
<td>Easily supports delegation (i.e., share key)</td>
<td>Easily supports centralized global security policy</td>
</tr>
<tr>
<td>Computationally expensive</td>
<td>Computationally cheap</td>
</tr>
<tr>
<td>Requires key management and distribution mechanisms</td>
<td>Requires policy management mechanisms</td>
</tr>
<tr>
<td><em>Requires key security</em></td>
<td><em>Requires reference monitor</em></td>
</tr>
</tbody>
</table>

**Linux has had broad crypto support since version 2.5, primarily for verifying signatures...**

...*but in the kernel, access control is king!*
Access Control

• Determine whether a principal can perform a requested operation on a target object

  • **Principal/Subject**: user, process, etc.
  • **Operation/Action**: read, write, etc.
  • **Object**: file, tuple, etc.
Protection Domains

- A computer system is a set of processes and objects
- A process operates within a protection domain
- A protection domain specifies the resources a process may access and the types of operations that may be invoked on the objects.

- **The Principle of Least Privilege:** The protection domain of a process should be as small as possible given the need of that process to accomplish its assigned task.
Example of 3 protection domains:

Domain 1:
- File1[R]
- File2[RW]

Domain 2:
- File3[R]
- File4[RWX]
- File5[RW]
- Printer1[W]

Domain 3:
- File6[RWX]
- Plotter2[W]

What are domains based on in Linux?
Discretionary Access Control (DAC)

• Owner of creator of resources specify which subjects have which access to those resources

• Commonly implemented in commercial products

• Access is managed by individual users, not a central security policy

Find the access control!
Access Mask defines permissions for User, Group, and Other.

```
chmod u=rwx,g=rx,o=r myfile
chmod 754 myfile
```

4 stands for "read",
2 stands for "write",
1 stands for "execute", and
0 stands for "no permission."
Access Control Lists (ACLs)

• Each column in access matrix specifies access for one Object.

• On invocation of a method $R$ on an object $O$ by a process running in a domain $D$…
  
  • … the access control list is searched to check whether $D$ is allowed to perform method $R$ on object $O$ (e.g., allowed to read the file or execute the program)

• A default (e.g., “rest of the world”) can be associated with an access list so that any Domain not specified in the list can access the Objects using default methods.

• It is easy for the owner of the Object to grant access to another Domain or revoke access.

• ACL entries can be for individual users or for a group of users.
Use of access control lists to manage file access in UNIX
Capabilities

- An alternative access control mechanism
- Capability is an unforgeable ticket
  - Managed by OS
  - Can be passed from one process to another
- Permissive
- OS Mechanism (Reference monitor) checks ticket
  - Does not need to know the identity of the user/proc
- Can be used to partition superuser privilege
- Implementation: POSIX.1e capabilities
When capabilities are used, each process has a capability list.
Problems?

• What might go wrong with DAC or Capabilities?
  • Security is left to the discretion of subjects
  • Impossible to guarantee security of system
  • Security of system changes over time.

• Solution?
  • **Mandatory Access Control**: Operating system constrains the ability of subjects (even owners) to perform operations on objects according to a system-wide security policy.
- A multi-level security model that provides strong confidentiality guarantees.
- Formalizes Classified Information
- State machine (Lattice) specifies permissible actions
- Lattice is comprised of both *levels* and *categories*
• **The Simple Security Property**: A subject running at security level k can read only objects at its level or lower. (*no read up*)

• **The * Property**: A subject at security level k can write only objects at its level or higher (*no write down*)
Using Bell-Lapadula, we can reason about permissible information flows in a system.
• Bell-LaPadula provides confidentiality. What about integrity?

• Biba model provides Integrity guarantees in a manner analogous to Bell-Lapadula’s secrecy levels.

• Integrity prevents inappropriate modification of data.
• **The Simple Integrity Property:** A subject running at integrity level $k$ must not read an object at a lower integrity level (*no read down*)

• **The * Integrity Property:** A subject at security level $k$ can only write objects at its level or lower. (*no write up*)
Combining Security Assurances

- **What happens when we want both confidentiality and integrity?**
  - i.e., we deploy Bell-Lapadula and Biba at the same time?
- All flows of information across levels “shut down”
- In practice, we need ways for data to change levels
  - e.g., in Clark-Wilson Integrity, Transformation Procedures (TPs) take low integrity data as input and output high integrity data.
  - Analogous to system call gates that exist between user space and kernel space!
All the Access Controls

• Basic Access Matrix
  • UNIX, ACL, various capability systems
• Aggregated Access Matrix
  • Type Enforcement (TE), Role-Based (RBAC), groups and attributes, parameterized
• Plus Domain Transitions
  • DTE, SELinux, Java
• Lattice Access Control Models
  • Bell-LaPadula, Biba, Denning
• Predicate Models
  • ASL, OASIS, Clark-Wilson, domain-specific models, many others
• Safety Models
  • Take-grant, Schematic Protection Model, Typed Access Matrix
Cool. But how do we implement these models in an operating system?
• Where to make access control decisions? (Mediation)
• Which access control decisions to make? (Authorization)
• Decision function: Compute decision based on request and the active security policy

• **Reference Monitor Concept (i.e., Goals):**
  • Complete Mediation
  • Tamper Proof
  • Verifiable