Goals for Today

• **Learning Objective:**
  • Discover how mandatory access controls operate in modern operating systems

• **Announcements, etc:**
  • MP4 due April 24th
  • C4 Students: Presentations start this Wednesday!

**Reminder**: Please put away devices at the start of class
All of security is summarized in the “CIA” Triad:

1. **Confidentiality**: Hide information from those subjects unauthorized to view it.
2. **Integrity**: Ensure data is an accurate and unchanged representation of the original secure information.
3. **Availability**: Ensure that data is accessible to authorized subjects at all times.
Problems w/ DAC?

• 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.
Multi-Level Security (MLS)

- **Bell-Lapadula for Confidentiality**
  - No Read Up, No Write Down
- **Biba for Integrity**
  - No Read Down, No Write Up
Using Bell-Lapadula, we can reason about permissible information flows in a system.
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 Goals:**

- Complete Mediation
- Tamper Proof
- Verifiable
Ok, back to MLS…

• Reference Monitors enable enforcement of arbitrary MAC security policies. So time to deploy B-LP-Linux?

• In practice, lots of information flows violate MLS
  • Example: passwd usage features IF violation...

• Bell-LaPadula solution to resolving information flow violations: manual intervention!
  • OK for Government secrets
  • Not great for computer systems

• So, what do we do?
Solutions for access control are plentiful…

- **Basic Access Matrix**
  - UNIX, ACL, various capability systems
- **Aggregated Access Matrix**
  - TE, RBAC, groups and attributes, parameterized
- **Plus Domain Transitions**
  - DTE, SELinux, Java
- **Lattice Access Control Models**
  - Bell-LaPadula, Biba, Denning
- **Predicate Models**
  - ASL, OASIS, domain-specific models, many others
- **Safety Models**
  - Take-grant, Schematic Protection Model, Typed Access Matrix
SELinux

• Designed by the NSA
• A more flexible solution than MLS
• SELinux Policies are comprised of 3 components:
  • **Labeling State** defines security contexts for every file (object) and user (subject).
  • **Protection State** defines the permitted <subject,object,operation> tuples.
  • **Transition State** permits ways for subjects and objects to move between security contexts.
• Files and users on the system at boot-time must be associated with their security labels (contexts)
  • Map file paths to labels via regular expressions
  • Map users to labels by name
    • User labels pass on to their initial processes
• How are new files labeled? Processes?
MAC Policy based on *Type Enforcement*

Access Matrix Policy

- Processes with subject label…
- Can access file of object label
- If operations in matrix cell allow

Focus: Least Privilege

- Only provide permissions necessary

<table>
<thead>
<tr>
<th></th>
<th>O₁</th>
<th>O₂</th>
<th>O₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₁</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>S₂</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>S₃</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
• Permissions in SELinux can be produced with runtime analysis.

• **Step 1**: Run programs in a controlled (no attacker) environment without any enforcement.

• **Step 2**: Audit all of the permissions used during normal operation.

• **Step 3**: Generate policy file description
  • Assign subject and object labels associated with program
  • Encode all permissions used into access rules
• Premise: Processes don’t need to run in the same protection state all of the time.

• Borrows concepts from Role-Based Access Control

• Example: `passwd` starts in user context, transitions to privileged context to update `/etc/passwd`, transitions back to user.
Include SELinux in Linux 2.5!
Include SELinux in Linux 2.5!

I’m just not that into you...
Linux Security circa 2000

- Patches to the Linux kernel
  - Enforce different access control policy
    - Restrict root processes
  - Some hardening
- Argus PitBull
  - Limited permissions for root services
- RSBAC
  - MAC enforcement and virus scanning
- grsecurity
  - RBAC MAC system
  - Auditing, buffer overflow prevention, /tmp race protection, etc
- LIDS
  - MAC system for root confinement
Linus’s Dilemma

SELinux, DTE, MAC, …hmmmm

What is the right solution?
Linus’s Dilemma

The answer to all computer science problems…
add another layer of abstraction!
“to allow Linux to support a variety of security models, so that security developers don't have to have the ‘my dog's bigger than your dog’ argument, and users can choose the security model that suits their needs.”, Crispin Cowan

Linux Security Modules

Before LSM

Access control models implemented as Kernel patches

After LSM

Access control models implemented as Loadable Kernel Modules
LSM Requirements

- LSM needs to reach a balance between kernel developer and security developers requirements. LSM needs to unify the functional needs of as many security projects as possible, while minimizing the impact on the Linux kernel.
  - Truly generic
  - Conceptually simple
  - Minimally invasive
  - Efficient
  - Support for POSIX capabilities
  - Support the implementation of as many access control models as Loadable Kernel Modules
LSM Architecture

- Linux Kernel modified in 5 ways:
  - Opaque security fields added to certain kernel data structures
  - Security hook function calls inserted at various points with the kernel code
  - A generic security system call added
  - Function to allow modules to register and unregistered as security modules
  - Move capabilities logic into an optional security module
Opaque Security Fields

- Enable security modules to associate security information to Kernel objects

- Implemented as void* pointers

- Completely managed by security modules

- What to do about object created before the security module is loaded?
Security Hooks

- Function calls that can be overridden by security modules to manage security fields and mediate access to Kernel objects.

- Hooks called via function pointers stored in security->ops table

- Hooks are primarily “restrictive”
Security Hooks

```c
ssize_t vfs_read(...) {
    ...
    ret = security_file_permission(file, ...);
    if (!ret) {
        ret = file->f_op->read(file, ...);
    }
    ...
}
```
Security Hook Details

• Difference from discretionary controls
  – More object types
    • 29 different object types
    • Per packet, superblock, shared memory, processes
    • Different types of files
  – Finer-grained operations
    • File: ioctl, create, getattr, setattr, lock, append, unlink,
  – System labeling
    • Not dependent on user
  – Authorization and policy defined by module
    • Not defined by the kernel
LSM Hook Architecture

- **Syscall**
- **Load Policy**
- **Register/Unregister**

**Linux Kernel**

- **Hook**
- **SysFS**
- **Object Label**
- **LSM**

Define
Process open System Call

Lookup inode

Process file path down to inode (resolving directories/links)

DAC checks

LSM hook

"Is user_process allowed to perform operation on inode?"

LSM Policy Engine

User Space

Kernel Space

CS 423: Operating Systems Design
LSM Use

• Available in Linux 2.6
  – Packet-level controls upstreamed in 2.6.16

• Modules
  – POSIX Capabilities module
  – SELinux module
  – Domain and Type Enforcement
  – Openwall, includes grsecurity function
  – LIDS
  – AppArmor

• Not everyone is in favor
  – How does LSM impact system hardening?