Host Security

Peter Chapin
Vermont State University
CIS-3240, Computer Security

The Three As

- **A**uthentication
  - Determining the identity of a principal
- **A**uthorization
  - Determining what actions a previously authenticated principal can do
- **A**uditing
  - Tracking the activities of a principal

Logging vs Auditing

- Logging
  - Information saved by choice
    - Applications log their usage
    - People can log their activities
  - Record may be incomplete
    - Applications may not log all “interesting” things
    - People may forget to log an activity
  - Policy customizable for each application/activity
    - Web server logs all requests, etc.

Logging vs Auditing

- Auditing
  - Information saved by decree
    - For example, OS records all file I/O activity without application permission or control
  - More complete record
    - OS sees all file activity done by any program anywhere at any time
  - Policy more global
    - Fine-tuning the audit is a pain. Also, audits usually generate huge data sets that require special tools to manipulate

Log Modification

- Protect the log!
  - Attackers will try to modify it to cover themselves
  - Audit logs tend to be harder to access
  - Burn logs to read-only media (write-once DVD?)
- Paper logs?
  - Writing a log of activities on paper has advantages
    - Available when the machine isn't
    - Can't be modified by an attacker on the machine

Paper Log Recommendations

- Recommendations
  - Bound notebook
    - No loose-leaf paper!
    - No random scrap paper
  - No blank pages
  - Signature + witness signature at the end of each day
  - More convincing in court, should that be necessary
- Digital signatures might also work

Authorization

- Principal: The entity to be authorized
  - Often, a person who is known to the system
  - Could be a group or role
  - Could be a process or a device
- Object: The entity to which access is granted
  - Often, a digital object such as a file
  - Could be an abstract activity (“shutdown system”)
  - Could be a process or a device

Permissions

- Actions that can be taken
  - Precise set of permissions varies
    - **Principal Permission Object**
    - Alice can read afile.txt
    - Bob can transmit through serial port #1
    - Carol can not shutdown the system

Access Control Matrix

- A way of thinking about access control
  - One row for each principal in the system (Alice, Bob, Carol, …)
  - One column for each object in the system (afile.txt, serial port, …)
  - Each cell in the matrix contains the permissions the principal has over the object.
    - Example: At (*Alice*, *afile.txt*), we have (*read*)

Not Real

- The access control matrix is just a concept
  - Not usually practical to actually store all this information in a single place
- Helps to organize our understanding
  - Columns: Attach permissions to each object (***Access Control Lists***)
  - Rows: Attach permissions to each principal (***Capabilities***)

Access Control Lists

- File systems usually use this approach
  - Attach to each file a list of who can do what to that file.
- Because of the vast number of files, special abbreviations are often used
  - File permissions might “inherit” from the enclosing folder
  - Default permissions are applied to each new file.

Capabilities

- Capabilities used for abstract operations
  - Alice has the “shutdown the system” capability
- Capabilities often cut across multiple objects
  - The “configure network” capability might include:
    - The ability to install/remove network drivers
    - The ability to modify critical network-related files
    - The ability to view all packets on the network

Groups

- A Group is a special principal that contains other principals
  - Permissions assigned to the group apply to all group members
  - Major convenience in complex systems
    - Administrator changes group membership when people join/leave projects. Permissions don't need to be changed for individuals.
  - Hierarchy of groups? Groups within groups? Group inheritance?

Role Based Access Control

- RBAC: Grant permissions to *roles* and then put principals into the roles they need.
  - Example: Network Administrator role… has permission to install/remove network cards, etc. Alice is put into that role.
- RBAC vs Groups?
  - Not much different. Most RBAC systems allow for *role activation*. A principal can turn their roles off and on at will.
  - In contrast, a principal is usually always in all groups for which they are a member

Unix

- Traditional Unix systems are simple
  - One “super user” with total power. Ordinary users have no special power.
    - Disastrous if the superuser is compromised
    - A mistake by the superuser can trash the system
  - Simple group system with limited access control lists on files

Modern Unix

- Newer Unix systems fix the problems
  - Capabilities and RBAC allow the distribution of administrative power
    - Safer: the compromise of a sub-administrator account or an error by a user in a sub-administrator role causes only limited damage.
  - Richer access control list support (see “POSIX ACLs”)

Windows

- More modern design from the start
  - Very rich access control list features for files
    - Similar features in the Windows registry and in Active Directory
  - RBAC-like roles for operations.
  - Rich group/role organization (especially with Active Directory)

Is Your Windows System Secure?

- Who knows?
  - Remember: **Complexity is the enemy of security!**
  - What overall access does Alice have to a file after considering all of Alice's group and role memberships, inherited permissions, access control lists, and capabilities???
- This is a real problem in systems

Monotonic Access Control

- Permissions add access, never remove it
  - Example: Alice can **READ** afile.txt. Alice is in a group “Contributors” that can **WRITE** afile.txt. Thus, Alice can **READ/WRITE** file.txt.
  - Overall permission is the accumulation of parts
- If you want to be sure someone has a permission: grant it explicitly!
  - It doesn’t matter if they have (or not) the permission from elsewhere already

Negative Permission

- “Alice CAN NOT read afile.txt”
  - Applying a negative permission forces the permission’s removal
    - Regardless of whether Alice has it from somewhere else or not
  - Windows has this feature; many systems do not.
- Feature causes difficulties
  - Computing access is harder (maybe undecidable)
  - Computing access is slower
  - Computing access requires **total knowledge**

Total Knowledge?

- Suppose…
  - We have monotonic permissions and…
  - Alice shows you that she CAN read afile.txt
  - You are done. Grant access.
- Now…
  - If negative permissions are possible, you must…
  - Search the universe for all information about Alice to see if somewhere there is a CAN NOT permission about her. ***Impossible in a distributed environment!***

Discretionary Access Control (DAC)

- *Defn: The owner of an object sets permissions*
  - This is the “normal” model most people use
    - You can set permissions on your own files
  - Low security!
    - Programs executing as you can modify file permissions
    - … accidentally (or maliciously) grant access to others!
      - You execute a malicious attachment
      - You install a Trojan horse program

Mandatory Access Control (MAC)

- *Defn: The sys admin sets all permissions*
  - You can't change the permissions on your stuff!
    - Not usually used in casual environments.
    - More typical in high-security environments
  - High Security!
    - Programs executing as you can't access random things
      - Malicious attachments can't necessarily read your files
      - Trojan horse programs can't necessarily install a back door
  - Much larger burden on system administration!

Multi-Layer Security

- MLS often used by military
  - "Top Secret", "Secret", "Confidential", "Public"
  - In general, we can number the levels
    - Level 0 .. N-1, where level N-1 is the "highest" level
  - Each object is given a security level
  - Each principal acts at a particular security level
  - Let's use processes as principles
    - *Programs do things, people only run programs!*

Bell-LaPadula Model

- Confidentiality in an MLS system
  - *Simple Security Property*
    - A process at level K can only read objects at K or lower
  - **-property*
    - A process at level K can only write objects at K or higher
  - **Read Down/Write Up**
    - A process at K (a private) can reveal all it knows to a process at K + 1 (a general), but a process at K + 1 can't disclose information to a process at K.

Biba Model

- Integrity in an MLS system
  - *Simple Integrity Property*
    - A process at level K can only write objects at K or lower
  - *Integrity *-property*
    - A process at level K can only read objects at K or higher
  - **Write Down/Read Up**
    - A process at K (a private) can be told what to do by a process at K + 1 (a general), but a process at K + 1 can't read bogus information from a process at K.

Contradictory?

- Clearly, both models can't be followed exactly!
  - … otherwise, there would be no exchange of information across levels
  - *Insight: Any flow of information between security levels is a potential security problem! Minimize it!*
  - *Insight: Confidentiality and Data Integrity are duals (note dual roles of read/write!)*
  - *Insight: IPC involves read/write between processes*

Linux Security Modules

- LSM… Kernel modules that implement security features in the OS
  - Various MAC supporting modules have been made
    - Security Enhanced Linux (SELinux), designed by the NSA
      - Uses object labels and a policy database
      - Hard to configure and maintain
    - AppArmor
      - Uses path-based policy (instead of object-based)
      - Has a learning mode
    - Others…