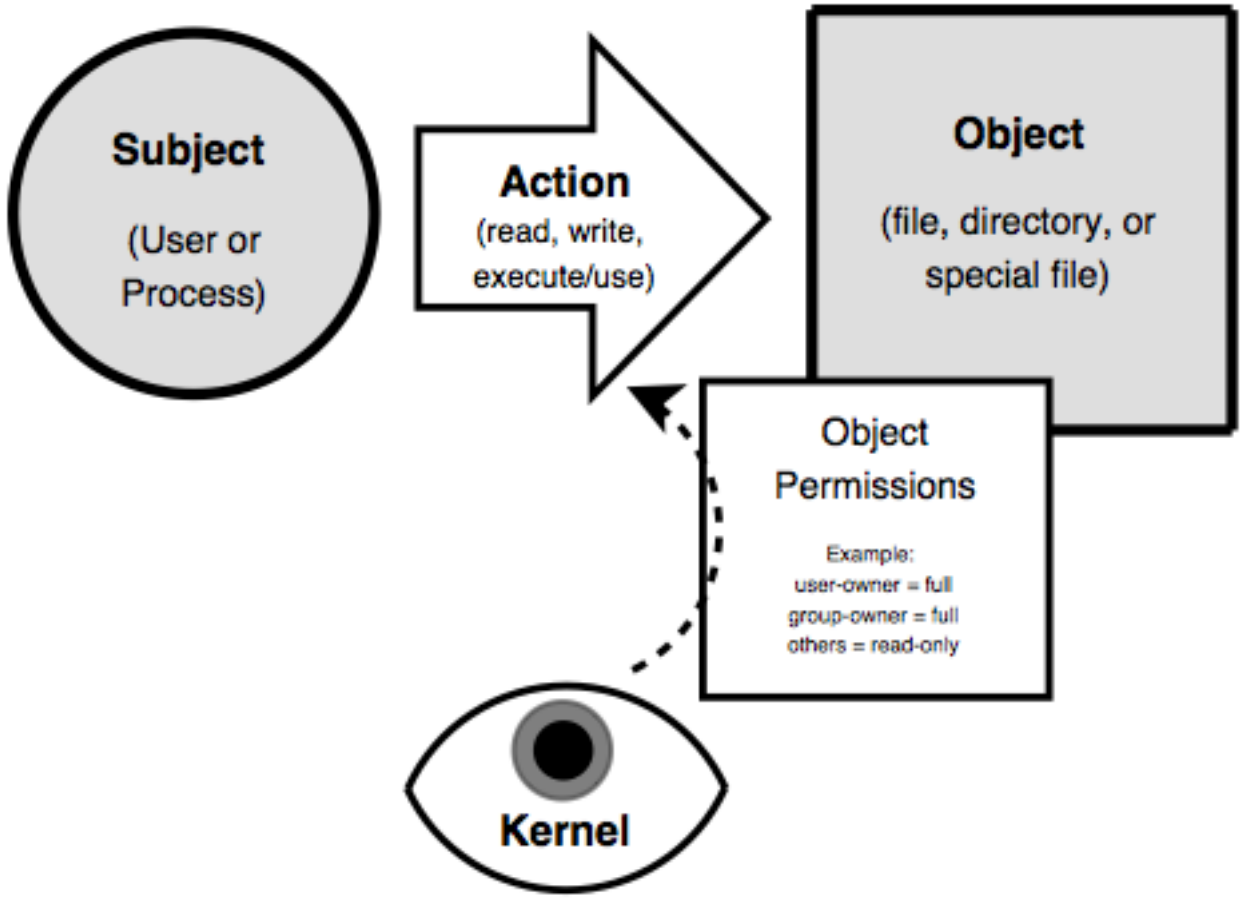
IA 313: Operating Systems Security

Lecture 4: Linux Security

Linux Security Model

- Linux's traditional security model is:
  - people or proceses with "root" privileges can do anything
  - other accounts can do much less
- Hence attacker's want to get root privileges
- Can run robust, secure Linux systems
- Crux of problem is use of **Discretionary Access Controls** (DAC)

Linux Security Transactions



File System Security

- In Linux *everything* is a file
  - e.g. memory, device-drivers, named pipes, and other system resources
  - hence why file system security is so important
- I/O to devices is via a “special” file
  - e.g. `/dev/cdrom`
  - i.e /dev/cdrom points to /dev/hdb which is a special file
- Have other special files like named pipes
  - a conduit between processes / programs

Users and Groups

- A user-account (user)
  - represents someone capable of using files
  - associated both with humans and processes
- A group-account (group)
  - is a list of user-accounts
  - users have a main group
  - may also belong to other groups
- Users & groups are **not** files

Users and Groups …

- User's details are kept in `/etc/password`

```
maestro:x:200:100:Maestro Edward
  Hizzersands:/home/maestro:/bin/bash
```

- Additional group details in `/etc/group`

```
conductors:x:100:

pianists:x:102:maestro,volodya
```

- Use **useradd**, **usermod**, **userdel** to alter

File Permissions

- Files have two owners: a user & a group
  - Each with its own set of permissions
  - With a third set of permissions for other
- Permissions are to read/write/execute in order user/group/other

```
-rw-rw-r--  1  maestro user 35414
  Mar 25 01:38 baton.txt
```

- Permission can be changed using chmod command

Directory Permissions

- read = list contents
- write = create or delete files in directory
- execute = use anything in or change working directory to this directory
- e.g.

```
$ chmod g+rx extreme_casseroles
$ ls -l extreme_casseroles
 drwxr-x--- 8 biff  drummers 288
 Mar 25 01:38 extreme_casseroles
```

Difference between File and Directory Permissions

Access Type	File	Directory
Read	If the file contents can be read	If the directory listing can be obtained
Write	If user or process can write to the file (change its contents)	If user or process can change directory contents somehow: create new or delete existing files in the directory or rename files.
Execute	If the file can be executed	If user or process can access the directory, that is, go to it (make it to be the current working directory)

Sticky Bit

- Originally used to lock file in memory
- Now used on directories to limit delete
  - if set must own file or dir to delete
  - other users cannot delete even if have write
- Set using chmod command with +t flag, e.g.

```
chmod +t extreme_casseroles
```

- Directory listing includes t or T flag

```
drwxrwx--T  8  biff  drummers  288  Mar 25
  01:38 extreme_casseroles
```

- Only apply to specific directory not child dirs

SetUID and SetGID

- SetUID bit: means program “run with same privileges as” owner
  - No matter who executes it
- SetGID bit: means “run with same privileges as” a member of group that owns it
  - Again regardless of who executes it
- *Are very dangerous* if set on file owned by root or other privileged account or group
  - Only used on executable files, not shell scripts
  - The command “sudo” is much better tool for delegating root’s authority
- Note that the linux kernel ignores the setid and setgid bits on shell scripts; these bits only work on binary (compiled) executables

SetGID and Directories

- SetUID has no effects on directories
- SetGID does and causes any file created in a directory to inherit the directory's group-owner
- Useful if users belong to other groups and routinely create files to be shared with other members of those groups
  - Instead of manually changing its group

Kernel Space and User Space

- Kernel space: refers to memory used by the Linux kernel and its loadable modules (e.g device drivers)
- User space: refers to memory used by all other processes
- Since kernel enforces Linux DAC and security, its extremely critical to isolate kernel from user space
  - For this reason, kernel space never swapped to disk
  - Only root may load and unload kernel modules

Linux Design Flaws/Poor Design Decisions

- Linux is based on a long history of well fleshed-out multi-user design
- Linux is mostly modular by design
- Linux does not depend upon RPC to function, and services are usually configured not to use RPC by default
- Linux servers are ideal for headless non-local administration

Based on Multi-User Design

- Linux does not have a history of being a single-user system
  - Therefore it has been designed from the ground-up to isolate users from applications, files and directories that affect the entire operating system
- Each user is given a user directory where all of the user's data files and configuration files are stored
  - When a user runs an application, such as a word processor, that word processor runs with the restricted privileges of the user

Modular by Design, not Monolithic

- Linux is for the most part a modularly designed operating system
  - From the kernel (the core “brains” of Linux) to the applications
- Not everything in Linux is modular
  - The two most popular graphical desktops: KDE and GNOME, are somewhat monolithic by design
- The Linux kernel supports modular drivers, but it is essentially a monolithic kernel where services in the kernel are interdependent
  - Any adverse impact of this monolithic approach is minimized by the fact that the Linux kernel is designed to be as minimal a part of the system as possible

Not Constrained by an RPC Model

- Most Linux distributions install programs with network access turned off by default
- Even when Linux applications use the network by default, they are most often configured to respond only to the local machine and ignore any requests from other machines on the network
- Unlike Windows Server 2003, you can disable virtually all network-related RPC services on a Linux machine and still have a perfectly functional desktop

Ideal for Headless Non-local Administration

- A Linux server can be installed, and often should be installed as a “headless” system (no monitor is connected) and administered remotely
  - Often the ideal type of installation for servers because a remotely administered server is not exposed to the same risks as a locally administered server
- This may be one of the most important differentiating factors between Linux and Windows
  - Because it virtually negates most of the critical security vulnerabilities that are common to both Linux and Windows systems, such as the vulnerabilities of the Mozilla browser vs. the Internet Explorer browser

Setuid root Vulnerabilities

- A **setuid root** program runs as root
  - *no matter who executes it*
- Used to provide unprivileged users with access to privileged resources (e.g. change passwd)
- Must be very carefully programmed
- If can be exploited due to a software bug
  - may allow otherwise-unprivileged users to use it to wield unauthorized root privileges
- Distributions now minimize setuid-root programs
- System attackers still scan for them!

Web Vulnerabilities

- A very broad category of vulnerabilities
  - because of ubiquity of world wide web have big and visible attack surfaces
- when written in scripting languages
  - not as prone to classic buffer overflows
  - can suffer from poor input-handling
- Few “enabled-by-default” web applications
- But users install vulnerable web applications or write custom web applications having easily-identified and easily-exploited flaws

Rootkits

- Allow attacker to cover their tracks
- If successfully installed before detection, all is very nearly lost
- Originally collections of hacked commands
  - hiding attacker's files, directories, processes
- Now use loadable kernel modules
  - intercepting system calls in kernel-space
  - hiding attacker from standard commands
- May be able to detect with chkrootkit
- Generally have to wipe and rebuild system

OS Installation

- Security begins with O/S installation
- Especially what software is run
  - since unused applications liable to be left in default, un-hardened and un-patched state
- Generally should not run:
  - X Window system, RPC services, R-services, inetd, SMTP daemons, telnet etc
- Also have some initial system s/w configuration:
  - setting root password
  - creating a non-root user account
  - setting an overall system security level
  - enabling a simple host-based firewall policy
  - enabling SELinux

Patch Management

- Installed server applications must be:
  - configured securely
  - kept up to date with security patches
- Patching can never win “patch rat-race”
- Have tools to automatically download and install security updates
  - e.g. up2date, YaST, apt-get
  - Note: should not run automatic updates on change-controlled systems without testing

Network Access Controls

- Network a key attack vector to secure
- TCP wrappers a key tool to check access
  - originally tcpd inetd wrapper daemon
  - before allowing connection to service checks
    - if requesting host explicitly in hosts.allow is ok
    - if requesting host explicitly in hosts.deny is blocked
    - if not in either is ok
  - checks on service, source IP, username
  - now often part of app using libwrappers

Network Access Controls …

- Also have the very powerful **netfilter** Linux kernel native firewall mechanism
  - and **iptables** user-space front end
- As useful on firewalls, servers, desktops
- Direct config tricky, steep learning curve
- Do have automated rule generators
- Typically for “personal” firewall use will:
  - allow incoming requests to specified services
  - block all other inbound service requests
  - allow all outbound (locally-originating) requests
- if need greater security, manually config

Antivirus Software

- Historically Linux is not vulnerable to viruses
- prompt patching was effective for worms
- but viruses abuse users privileges
- non-root users have less scope to exploit
  - but can still consume resources
- growing Linux popularity mean exploits
- hence antivirus software will be more important
  - various commercial and free Linux A/V

User Management

- guiding principles in user-account security:
  - need care setting file / directory permissions
  - use groups to differentiate between roles
  - use extreme care in granting / using root privs
- commands: chmod, useradd/mod/del, groupadd/mod/del, passwd, chage
- info in files /etc/passwd & /etc/group
- manage user's group memberships
- set appropriate password ages

Root Delegation

- Have "root can do anything, users do little" issue
- "su" command allows users to run as root
  - either root shell or single command
  - must supply root password
  - means likely too many people know this
- SELinux RBAC can limit root authority, complex
- "sudo" allows users to run as root
  - but only need their password, not root password
  - /etc/sudoers file specifies what commands allowed
- or configure user/group perms to allow, tricky

Logging

- Effective logging a key resource
- Linux logs using syslogd or Syslog-NG
  - receive log data from a variety of sources
  - sorts by **facility** (category) and **severity**
  - writes log messages to local/remote log files
- Syslog-NG preferable because it has:
  - variety of log-data sources / destinations
  - much more flexible “rules engine” to configure
  - can log via TCP which can be encrypted

Log Management

- Applications can usually be configured to log to any level of detail (debug to none)
- Need appropriate setting
- Must decide if use dedicated file or system logging facility (e.g. syslog)
  - central facility useful for consistent use
- Manage size of log files
  - must rotate log files and delete old copies
  - typically use logrotate utility run by cron
  - to manage both system and application logs

Running As Unprivileged User/Group

- every process “runs as” some user
- extremely important this user is not root
  - since any bug can compromise entire system
- may need root privileges, e.g. bind port
  - have root parent perform privileged function
  - but main service from unprivileged child
- user/group used should be dedicated
  - easier to identify source of log messages

Running in chroot Jail

- chroot confines a process to a subset of /
  - maps a virtual “/” to some other directory
  - useful if have a daemon that should only access a portion of the file system, e.g. FTP
  - directories outside the chroot jail aren’t visible or reachable at all
- contains effects of compromised daemon
- complex to configure and troubleshoot
  - must mirror portions of system in chroot jail

Modularity

- Applications running as a single, large, multipurpose process can be:
  - more difficult to run as an unprivileged user
  - harder to locate / fix security bugs in source
  - harder to disable unnecessary functionality
- Hence modularity a highly prized feature
  - providing a much smaller attack surface
- cf. postfix vs sendmail, Apache modules

Encryption

- Sending logins & passwords or application data over networks in clear text exposes them to network eavesdropping attacks
- Hence many network applications now support encryption to protect such data
  - often using OpenSSL library
- May need own X.509 certificates to use
  - can generate/sign using openssl command
  - may use commercial/own/free CA

Mandatory Access Controls

- Linux uses a DAC security model
- But Mandatory Access Controls (MAC) impose a global security policy on all users
  - users may not set controls weaker than policy
  - normal admin done with accounts without authority to change the global security policy
  - but MAC systems have been hard to manage
- Novell's SuSE Linux has AppArmor
- RedHat Enterprise Linux has SELinux
- Pure SELinux for high-sensitivity, high-security

SELinux

- Is NSA's powerful implementation of mandatory access controls for Linux
- Linux DACs still applies, but if it allows the action SELinux then evaluates it against its own security policies
- “Subjects" are processes (run user cmds)
- Actions are "permissions”
- Objects not just files & dirs
- To manage complexity SELinux has:
  - "that which is not expressly permitted, is denied”
  - groups of subjects, permissions, and objects

Security Contexts

- Each individual subject & object in SELinux is governed by a **security context** being a:
  - user - individual user (human or daemon)
    - SELinux maintains its own list of users
    - user labels on subjects specify account's privileges
    - user labels on objects specify its owner
  - role - like a group, assumed by users
    - a user may only assume one role at a time,
    - may only switch roles if and when authorized to do so
  - domain (type) - a sandbox being a combination of subjects and objects that may interact with each other
- This model is called **Type Enforcement** (TE)

Decision Making in SELinux

- Two types of decisions:
- **Access** decisions
  - when subjects do things to objects that already exist, or create new things in expected domain
- **Transition** decisions
  - invocation of processes in different domains than the one in which the subject-process is running
  - creation of objects in different types (domains) than their parent directories
  - transitions must be authorized by SELinux policy

RBAC and MLS Controls

- Have **Role Based Access Control** (RBAC)
  - rules specify **roles** a user may assume
  - other rules specify circumstances when a user may **transition** from one role to another
- and **Multi Level Security** (MLS)
  - concerns handling of classified data
    - “no read up, no write down”
  - MLS is enforced via file system labeling

SELinux Policy Management

- Creating and maintaining SELinux policies is complicated and time-consuming
- A single SELinux policy may consist of hundreds of lines of text
- RHEL has a default “targeted” policy
  - defines types for selected network apps
  - allows everything else to use DAC controls
- Have a range of SELinux commands

Novell AppArmor

- Novell's MAC for SuSE Linux
  - enforced at kernel level
  - using Linux Security Modules
- Restricts behavior of selected applications in a very granular but targeted way
  - hence a compromised root application's access will be contained
  - has no controls addressing data classification
  - hence only a partial MAC implementation
- Non-protected apps just use Linux DAC