Computer Security: Principles and Practice

Chapter 23 – Linux Security

EECS 710

Professor: Dr. Hossein Saiedian

Presented by Ankit Agarwal

Outline

- Introduction
- Linux Security Model
- Linux File-System Security
- Linux Vulnerabilities
- Linux System Hardening
- Application Security
- Mandatory Access Controls

Introduction

- Linux – Unix like computer OS that uses Linux kernel
- created by Linus Torvalds in 1991
- evolved into a popular alternative to Win and MAC OS
- has many features and applications
  - desktop and server OS, embedded systems
  - hence wide variety of attacks possible
  - various security tools available
- it uses Discretionary Access Control Model
- Mandatory Access Controls implemented
  - to make up for DAC shortcomings
  - SELinux and Novell AppArmor

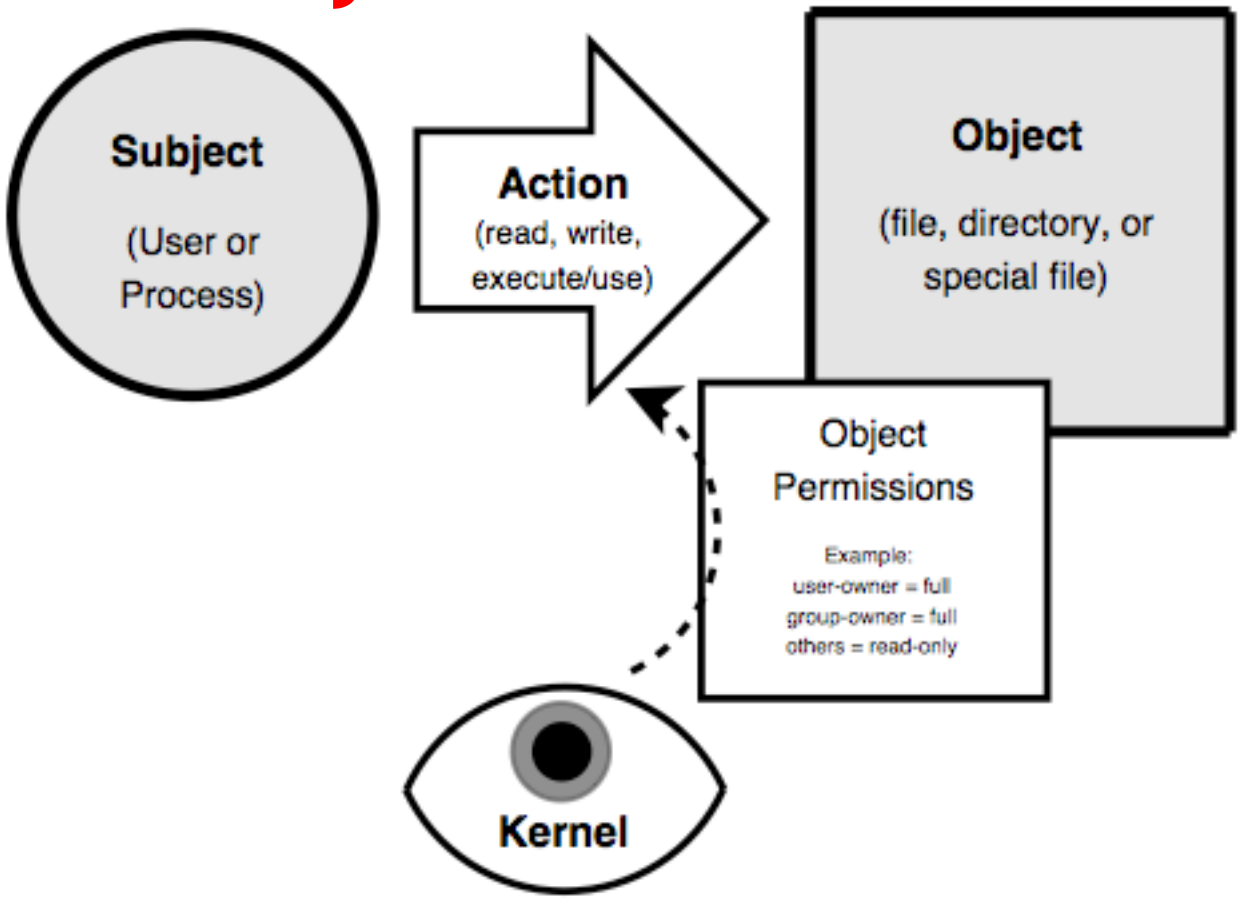
Outline

- Introduction
- Linux Security Model
- Linux File-System Security
- Linux Vulnerabilities
- Linux System Hardening
- Application Security
- Mandatory Access Controls

Linux Security Model

- Traditional security model
  - people or processes with “root” privileges can do anything
  - other accounts can do much less
- Goal of hackers – to gain root privilege
- Linux can be run robust and secure
  - many system admins. fail to use the security features
  - add-on tools like sudo and Tripwire available
- Crux of the problem – Discretionary Access Control

Linux Security Transactions



Outline

- Introduction
- Linux Security Model
- Linux File-System Security
- Linux Vulnerabilities
- Linux System Hardening
- Application Security
- Mandatory Access Controls

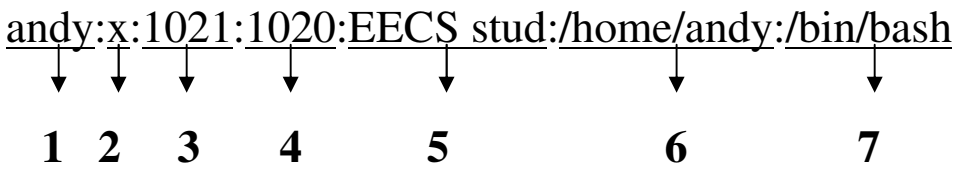
Linux File System

- In Linux everything is a file
- I/O to devices is via a “special” file
  - e.g. /dev/cdrom points to /dev/hdb which is a special file
- have other special files like named pipes
  - a conduit between processes / programs
- since almost everything a file – security very important

Users and Groups

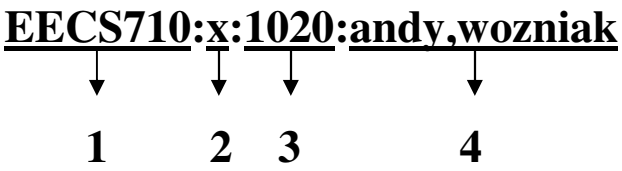
- Users and Groups are not files
- users
  - someone or something capable of using files
  - can be human or process
  - e.g. lpd (Linux Printer Daemon) runs as user lp
- groups
  - list of user accounts
  - user's main group membership specified in /etc/passwd
  - user can be added to additional group by editing /etc/group
  - command line -> useradd, usermod, and userdel

Understanding /etc/passwd



1. **username**: Used when user logs in. It should be between 1 and 32 characters in length.
2. **password**: An x character indicates that encrypted password is stored in /etc/shadow file.
3. **user ID (UID)**: Each user must be assigned a user ID (UID). UID 0 (zero) is reserved for root and UIDs 1-99 are reserved for other predefined accounts. UID 100-999 are reserved by system for administrative and system accounts/groups.
4. **group ID (GID)**: The primary group ID (stored in /etc/group file)
5. **user ID Info**: The comment field. Allows you to add extra information about the users such as user's full name, phone # etc. This field used by finger command.
6. **home directory**: The absolute path to the directory the user will be in when they log in. If this directory does not exists then users directory becomes /
7. **command/shell**: The absolute path of a command or shell (/bin/bash). Typically, this is a shell. Please note that it does not have to be a shell.

Snapshot of /etc/group



1. **group_name:** Name of group
2. **password:** Generally password not used, hence it is empty/blank. It can store encrypted password. Useful to implement privileged groups
3. **group ID (GID)**: Group ID must be assigned to every user
4. **group List**: List of user names of users who are members of the group. The user names must be separated by commas

File Permissions

- every file or folder in Linux has three types of access permissions
  - read (r), write (w), execute (x) access
- permission defined by three types of users
  - owner of file, group that owner belongs to, others

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

(Example from text)

- command line -> chmod

Directory Permissions

- permissions on folders work slightly differently

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


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

- used to trigger process to “stick” in memory or lock file in memory
  - usage now obsolete
- currently used on directories to suppress deletion of file that is owned by others
  - other users cannot delete even if they have write permissions
- chmod command with +t flag, e.g.

```
chmod +t extreme_casseroles (Example from text)
```

- directory listing includes t or T flag

```
drwxrwx--T  8  biff  drummers  288  Mar 25 01:38
  extreme_casseroles          (Example from text)
```

- the permissions are not inherited by child directories

SetUID and SetGID

- setuid bit means file when executed runs with the same permissions as the owner of the file
- setgid bit means file when executed runs as a member of the group which owns it
- *are very dangerous* if set on file owned by root or other privileged account or group
  - only used on executable files, not on shell scripts

SetGID and Directories

- setuid has no effect on directories
- setgid does and causes any file created in a directory to inherit the directory's group
- useful if users belong to other groups and routinely create files to be shared with other members of those groups

Numeric File Permissions

- read (r) = 4
- write (w) = 2
- execute (x) = 1

```
drwxr-x---  8 biff  drummers 288
Mar 25 01:38 extreme_casseroles
```

(Example from text)

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, important to isolate kernel from user
  - so kernel space never swapped to disk
  - only root may load and unload kernel modules

Linux Vulnerabilities

- Default Linux installations (unpatched and unsecured) have been vulnerable to
  - buffer overflows
  - race conditions
  - abuse of programs run “setuid root”
  - Denial of Service (DoS)
  - web application vulnerabilities
  - rootkit attacks

Outline

- Introduction
- Linux Security Model
- Linux File-System Security
- Linux Vulnerabilities
- Linux System Hardening
- Application Security
- Mandatory Access Controls

setuid root Vulnerabilities

- A **setuid root** program is a root-owned program
  - *runs as root no matter who executes it*
- unprivileged users can gain access to unauthorized privileged resources
- must be very carefully programmed
- setuid root programs necessary
  - e.g. to change password
- distributions now do not ship with unnecessary setuid-root programs
- system attackers still scan for them

Web Vulnerabilities

- a very broad category of vulnerabilities
- when written in scripting languages
  - not as prone to classic buffer overflows
  - can suffer from poor input-handling, XSS, SQL code injection etc.
- Linux distributions ship with few “enabled-by-default” web applications
  - E.g. default cgi scripts included with Apache Web server

Rootkits

- if successfully installed before detection, it is very difficult to find and remove
- originally began as collections of hacked commands
  - hiding attacker's files, directories, processes
- now use loadable kernel modules (LKMs)
  - intercepts system calls in kernel-space
  - hides attacker from user
- even LKMs not completely invisible
  - may be able to detect with chkrootkit
- generally have to wipe and rebuild system

Outline

- Introduction
- Linux Security Model
- Linux File-System Security
- Linux Vulnerabilities
- Linux System Hardening
- Application Security
- Mandatory Access Controls

Linux System Hardening

- this is done at OS and application level
- generalized steps to Linux System Hardening
  - preliminary Planning
  - physical System Security
  - operating System Installation
  - securing Local File Systems
  - configuring and Disabling Services
  - securing the root account
  - user Authentication and User Account Attributes
  - securing Remote Authentication
  - setup Ongoing System Monitoring
  - backups

OS Installation

- security begins with O/S installation
- what software is run
  - unused applications liable to be left in default, un-hardened and un-patched state
- generally should not run:
  - SMTP relay, X Window system, RPC services, R-services, inetd, SMTP daemons, telnet etc
- setting 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
  - should not run automatic updates on change-controlled systems without testing

Network Access Controls

- network a key attack vector to secure
- Libwrappers & TCP wrappers a key tool to check access
  - tcpd before allowing connection to service, checks
    - controls defined in /etc/hosts.allow
    - controls defined in /etc/hosts.deny
- using iptables for “Local Firewall” Rules
    - Use strong net filter commonly referred to as iptables
    - Inbuilt functionality in Linux

Antivirus Software

- historically Linux not as vulnerable to viruses
- windows targeted more due to popularity
- prompt patching of security holes more effective for worms
- viruses abuse users privileges
- non-privileged user account
  - less scope of being exploited
- growing Linux popularity means growing exploits
- hence antivirus software will be more important
  - various commercial and free Linux A/V

User Management

- guiding principles in user-account security:
  - be careful setting file / directory permissions
  - use groups to differentiate between roles
  - use extreme care in granting / using root privileges

- password aging
  - maximum and minimum lifetime for user passwords
  - globally changed in /etc/login.defs
  - to change password settings for existing users
    - command line -> change

Root Delegation

- "su" command allows users to run as root
  - use su with -c flag to allow you to run a command instead of an entire shell as root
  - must supply root password
  - drawback: many people will know root password
- SELinux RBAC can limit root authority but it's complex
- "sudo" allows users to run as root
  - but only need users password, not root password
  - sudoers defined in /etc/sudoers file
  - open and configure the sudoers file using 'visudo'

Logging

- Linux logs using syslogd or Syslog-NG
  - 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
- change default logging settings on both
- log files careful management
  - balance number and size of log files
  - rotate log files and delete old copies - logrotate

Outline

- Introduction
- Linux Security Model
- Linux File-System Security
- Linux Vulnerabilities
- Linux System Hardening
- Application Security
- Mandatory Access Controls

Application Security

- a large topic
- many security features are implemented in similar ways across different applications
- sub-topics covered
  - running as unprivileged user/group
  - running in chroot jail
  - modularity
  - encryption
  - logging

Running As Unprivileged User/Group

- every process "runs as" some user
- extremely important 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
  - directories outside the chroot jail aren't visible or reachable at all
- contains effects of compromised daemon
- complex to configure and troubleshoot

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 various network eavesdropping attacks
- hence many network applications now support encryption to protect such data
  - SSL and TLS protocols in OpenSSL library used
- may need own X.509 certificates to use
  - can generate/sign using openssl command
  - may use commercial/own/free CA

Logging

- applications can usually be configured to log to any level of detail (debug to none)
- centralized logging using (e.g. syslog) can be used for consistency
- must ensure there is some form of logging management as discussed before like rotating

Mandatory Access Controls

- Linux uses a DAC security model
- Mandatory Access Controls (MAC) imposes 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

Outline

- Introduction
- Linux Security Model
- Linux File-System Security
- Linux Vulnerabilities
- Linux System Hardening
- Application Security
- Mandatory Access Controls

SELinux

- is NSA's powerful implementation of MAC for Linux
  - Complicated – can be time-consuming to configure,troubleshoot
- Linux DACs still applies, but if it allows the action SELinux then evaluates it against its own security policies
- "subjects" are always processes (run user cmds)
- actions are called "permissions”
- objects not just files & directories include processes and other system resources
- SELinux manages complicacy by doing the following:
  - "that which is not expressly permitted, is denied”
  - by grouping 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
- Multi Level Security (MLS)
  - based on Bell-LaPadula (BLP) model
    - "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 run with only DAC controls
- have a range of SELinux commands
  - see references at end of chapter for details

Novell AppArmor

- Novell's MAC implementation for SuSE Linux
  - built on top of 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

Summary

- reviewed Linux security model and DAC
- Linux vulnerabilities
- Linux System Hardening
  - O/S and application hardening
- MAC, SELinux and AppArmor

Questions?

References

- Stallings, W., Brown, L., Computer Security: Principles and Practice, Upper Saddle River, NJ: Prentice Hall, 2008

- Unix System Hardening Checklist, Accessed Dec 8, 2008, http://www.linux-mag.com/downloads/2002-10/guru/harden_list.htm

- www.LinuxSecurity.com