Module 8
UNIX/Linux

Student Guide

Name: ______________________________ Class #: ______________

CIN: A-531-1900
Ver. 2020-01R4

Prepared for the U.S. Government by:

COMTECH COMMAND & CONTROL TECHNOLOGIES™

Table of Contents

Introduction

UNIX/Linux is an extremely versatile operating system (OS) that is portable to a variety of hardware platforms. As a CNO analyst, knowledge of the configuration and operation of the *Nix family of operating systems is essential. Linux is often the OS of choice for hackers and often used as the OS for web servers.

Safety/Hazard Awareness Notice

All personnel involved in operation and maintenance of electronic equipment must be thoroughly familiar with safety precautions as covered in Module 1.

Module Overview

This module, UNIX/Linux, utilizes concepts taught in earlier modules to expand students' knowledge and skills, providing them with tools needed to access the hidden powers of a Linux system. Linux is a challenging OS that provides great flexibility and control. Students are provided guidance and instruction on how to use native tools and concepts to modify the kernel, control system internals, and exploit and defend a Linux-based system and its associated networks. Students primarily work from a terminal using native commands to perform system enumeration tasks.

Module Testing Practices

This module consists of the following graded event requiring a 75% or better score.

- One performance test

This module also utilizes a minimum of one formal quiz and daily review quizzes in JCAC-eTC to reinforce the concepts required for obtaining a passing score on the above test.

Students are not to discuss tests/quizzes with one another. Test discussion outside of formal review with instructor is considered cheating. As a reminder, cheating is reported.

External resources are not permitted unless expressly directed by the instructor.

Resources

- Linux Pocket Guide

Module Objectives

Upon successful completion of this module, the student will:

8.1 Describe the objectives and architectural layers of a Linux OS.

8.2 Describe Linux kernel interactions with user processes, applications, and hardware.

8.3 Describe the Linux boot process.

8.4 Create, modify, and interpret shell scripts and their functions.

8.5 Manipulate and assess kernel modules and internal system components.

8.6 Configure system, service, and startup scripts.

8.7 Review and manipulate Linux system calls, functions, processes, security features and networking using system tools.

8.8 Describe file system fundamentals, inode management, and how data is stored and accessed.

8.9 Identify installed hardware components, disk-naming conventions, and file types.

8.10 Perform Linux system and network administration commands and evaluate results.

8.11 Modify user and group file permissions and ownerships.

8.12 Configure and use built-in software to allow file system interaction with other systems on the network.

8.13 Explain how Linux implements networking and how routing interfaces with LDAP and BIND protocols.

8.14 Create, modify, and manage system user accounts and associated files.

8.15 Utilize basic network administration tools and utilities (e.g., nmap and tcpwrappers) to configure and secure network services.

8.16 Describe and configure OS interoperability with NFS and Samba utilities.

8.17 Identify and evaluate patch and version level of web and email server security applications.

8.18 Perform and evaluate Linux system packet captures using built-in sniffers.

8.19 Assess the security posture of a Linux system using installed security scripts and configuration files such as console logins and banner grabbing.

8.20 Assess host security, configuration, and vulnerabilities using Firewalls, host-based Intrusion Detection Systems (IDSs), and banner grabbing techniques.

8.21 Configure Linux logging and scheduling facilities.

8.22 Perform file compression, un-compression, and archiving on a Linux system.

8.23 Assess system and network baseline deviations using system resources.

Objectives

Day 1

At the end of this training day, students will:

- Identify the advantages and fundamentals of a Linux kernel.
- Modify file and directory permissions.
- Modify file and directory ownership.
- View, find, and modify SUID, SGID, and sticky-bit permissions to achieve system security.
- Recall and utilize basic Linux commands.

Exercises

This training day includes the following exercises:

- Exercise 8-1, Permissions
- Exercise 8-2, Linux Command Refresher

JCAC-eTC

Complete self-study activities and assignments in the online training environment.

1 Linux Fundamentals

The Operating Systems (OS) module introduced the basic features and common commands of a Linux OS. This module provides a deeper familiarization of the Linux variant of UNIX and provides a basis for understanding its architectural layered approach. JCAC uses the Community Enterprise OS (CentOS) version of Linux on an X86 platform. Key advantages of the Linux kernel over other UNIX variants are:

- The Linux kernel is preemptive. The kernel can interrupt a task even as it is executing within the kernel.
- Linux supports the ability to dynamically load and unload kernel code (loadable kernel modules).
- Linux does not differentiate between threads and a normal process. All processes are the same but some share resources.
- Linux provides an object-oriented device model with device classes, hot-pluggable events, and a user-space device file system (*sysfs*).
- Linux is a free, open development model that ignores common UNIX features considered poorly designed by kernel developers.
- Linux is multi-user, multi-tasking, and portable to a variety of hardware platforms.

Linux system commands and file system fundamentals previously taught in the OS module remain relevant. Knowing key Linux directories, locations, and contents identified in the Linux File System Hierarchy Standard remain relevant and testable. **Error! Reference source not found.** lists previously taught commands to recall.

Table 1. Introductory Commands

cd	ls	pwd	less	more	tail	head
cat	file	mv	cp	ln	grep	find
man	mkdir	rmdir	rm	touch	vi	uname
tree	history	script	ps	kill	pkill	tar
w	who	whoami	chmod	useradd	userdel	echo
crontab	at	date	passwd	gzip	gunzip	clear
strings	su	sudo	which	|	>	>>

2 File Permissions

File permissions determines users authorized to access system files. Linux classifies everything as a file, including directories, devices, and even processes. Real files contain data and virtual files like those in */proc* contain system memory, mounted devices, and device configurations. Files may be world-writable, meaning anyone can write to them. Correctly setting permissions is vital to protecting a files content.

File permissions are set on the three classes of users listed below.

Owner	Actual owner of the file or directory, also referred to as *user*
Group	Group ownership of the file or directory
Other	All other users on the system, often referred to as "world"

The following are permission definitions for files and directories that apply to the classes.

Files	r	Read (view) file contents
	w	Write (modify) file contents
	x	Execute file as a program (i.e., script or executable)
Directories	r	Read (view) directory contents
	w	Write (create or delete) files or remove sub-directories within that parent directory.
	x	Execute allows users to change into directories using cd

2.1 Syntax of Permissions

The following example helps explain the syntax of file permissions:

```
[root@CentOS6_A ~]# ls -l /etc/nsswitch.conf
-rw-r--r--.   1  root   root    1796   Aug 18  07:59  nsswitch.conf
```

The permission field is divided into two sections: file type and file permissions (owner, group, other). The first character of the field indicates the type of file (-rw-r--r--).

-	Regular file
d	Directory
b	Block file
c	Character file
l	Symbolic link (points to another filename)
p	Named pipe
s	Socket

Characters 2 through 10 indicate file permissions (rw-r--r--).

r	Read
w	Write
x	Execute

- Characters 2 through 4 indicate the permissions of the owner (rw-------)
- Characters 5 through 7 indicate the permissions of the group (---r-----)
- Characters 8 through 10 indicate the permissions of other (------r--)

The link field follows and indicates the number of hard links associated with the file (1).

The owner field indicates the file owner (*root*).

The group field indicates the file group (*root*).

The size field indicates the file size in bytes (*1796*).

The date/time field indicates the change or modification date and time (*Aug 18 07:59*).

The file name field indicates the file name (*nsswitch.conf*).

The "**.**" following the permission set indicates a file with SELinux security context set. The Samba section discusses security context in detail.

2.2 Setting Permissions

Permissions for files and directories are set with the **chmod** command.

Syntax: **chmod** <octal/symbol> <file/directory name>

There are two methods to change permissions: numeric and symbolic.

Numeric: Uses octal code to specify three octal digits; one each for owner, group, and other. The below permissions would represent one of the three classes.

r	read	4
w	write	2
x	execute	1

```
[root@CentOS6_A ~]# chmod 744 /etc/testing

[root@CentOS6_A ~]# ls -l testing
-rwxr--r--   1  root   sys    Jun 27  10:59  test
```

Symbolic Uses alpha representation to assign permissions.

r	read
w	write
x	execute
u	(user) owner
g	group
o	other
+	add to existing permissions
-	removes permissions
=	set permissions exactly

```
[root@CentOS6_A ~]# chmod g+w,o=rx /etc/testing

[root@CentOS6_A ~]# ls -l testing
-rwxrw-r-x   1  root   sys   689 Aug 18 10:59 testing
```

By default, the system assigns the following permissions for newly created files and directories. The preceding ‘0’ in each of the following is explained in detail in the following section.

- 0666 File
- 0777 Directory

To modify the default permissions value of newly created files and directories use **umask**. Often called a command, utility, or program, **umask** is actually a shell built-in command. With no argument, **umask** displays the current value. If a **umask** is set to 0022, the binary weight of 2 (write) is subtracted from the *group* and *other* value of the default permission. The newly created file permissions become 0644 and newly created directory permissions become 0755.

To change user or group ownership, use the **chown** command.

Syntax: chown <options> user[:group] file(s)

To change the owner of a file:

```
[root@CentOS6_A tmp]# ls -l stufile.txt
-rw-r--r--. 1 instructor   army  0  Mar 21 13:28  stufile.txt

[root@CentOS6_A tmp]# chown student stufile.txt
-rw-r--r--. 1 student   army  0  Mar 21 13:35  stufile.txt
```

To change the group of a file:

```
[root@CentOS6_A tmp]# ls -l stufile.txt
-rw-r--r--. 1 student   army  0  Mar 21 13:37  stufile.txt

[root@CentOS6_A tmp]# chown :navy stufile.txt

[root@CentOS6_A tmp]# ls -l stufile.txt
-rw-r--r--. 1 student   navy  0  Mar 21 13:39  stufile.txt
```

To change both the owner and the group of a file:

```
[root@CentOS6_A tmp]# ls -l stufile.txt
-rw-r--r--. 1 student   navy  0  Mar 21 13:41  stufile.txt

[root@CentOS6_A tmp]# chown instructor:jcac stufile.txt

[root@CentOS6_A tmp]# ls -l stufile.txt
-rw-r--r--. 1 instructor   jcac  0  Mar 21 13:43  stufile.txt
```

Use the **-R** option to recursively set owner and group for all files in a directory structure, including sub-directories. The following is the result of changing the group ownership and the group of a file recursively.

```
[root@CentOS6_A tmp]# ls -l /home/student/flashcards.txt
-rw-r--r--. 1 student   navy  1209  Mar 21 13:44  flashcards.txt

[root@CentOS6_A tmp]# ls -l /home/student/testquiz.txt
-rw-r--r--. 1 student   navy  1723  Mar 21 13:48  testquiz.txt

[root@CentOS6_A ~]# chown -R :jcac /home/student

[root@CentOS6_A tmp]# ls -l /home/student/flashcards.txt
-rw-r--r--. 1 student   jcac  1209  Mar 21 13:44  flashcards.txt

[root@CentOS6_A tmp]# ls -l /home/student/testquiz.txt
-rw-r--r--. 1 student   jcac  1723  Mar 21 13:48  testquiz.txt
```

2.3 Set User and Group ID (SUID/SGID) Permissions

The Set User Identification/Set Group Identification (SUID/SGID) bits allow anyone to run an executable with the same privileges as the owner and/or group. The most common example is the **passwd** command. When a user's password is changed, a write to the */etc/shadow* file occurs. Only *root* user has *read* and *write* access. SUID/SGID is denoted by letter '*s*' in the owner/group permissions field, in place of the execution '*x*' bit.

```
[root@CentOS6_A ~]# ls -l /usr/bin/passwd
-rwsr-xr-x  1  root   root    25980  Feb 22 2012  /usr/bin/passwd
```

This permission set allows a process to have the same privileges as owner/group when a program executes. When a non-privileged user executes **passwd**, the effective UID of the process is *root,* the owner of the program, and not the user. The SUID privilege is used by **passwd** to modify the */etc/shadow* file.

Set SUID bit on a file with **chmod** using a `4' prefixed to the octal string.

```
[root@CentOS6_A ~]# ls -al  /test/run_me
-rwxr-xr-x   1  root  root  695  Jul 11 09:45 /test/run_me
[root@CentOS6_A ~]# chmod 4755 /test/run_me
[root@CentOS6_A ~]# ls -al /test/run_me
-rwsr-xr-x   1  root  root  695  Jul 11 09:45 /test/run_me
```

Set SGID bit on a file with **chmod** using a `2' prefixed to the octal string.

```
[root@CentOS6_A ~]# ls -al /test/run_me
-rwxr-xr-x  1  root  root  695   Jul 11 09:45 /test/run_me
[root@CentOS6_A ~]# chmod 2755 /test/run_me
[root@CentOS6_A ~]# ls -al /test/run_me
-rwxr-sr-x  1  root  root  695   Jul 11 09:45 /test/run_me
```

Set SUID and SGID bit with **chmod** using a `6' prefixed to the octal string.

```
[root@CentOS6_A ~]# ls -al /test/run_me
-rwxr-xr-x  1  root  root  695   Jul 11 09:45 /test/run_me
[root@CentOS6_A ~]# chmod 6755 /test/run_me
[root@CentOS6_A ~]# ls -al /test/run_me
-rwsr-sr-x  1  root  root  695   Jul 11 09:45 /test/run_me
```

Additionally, a "sticky-bit" may be applied to a directory that allows only the owner or root to delete or rename the directory. No other user has the privilege to delete the files created by some other user in that directory. The "sticky-bit" uses an octal value of "1" in the last field.

```
[root@CentOS6_A /]# ls -l
drwxrwxrwt  21  root root 4096 Jan 17 07:13 tmp
```

2.4 SUID/SGID Security

Setting SUID/SGID permissions on executable programs is relatively safe. However, setting these permissions on scripts may create a vulnerability. Because scripts parse commands sequentially, interrupting one during execution may produce a shell as the owner of the process running the command.

For example, a shell script written by *root* user allows users to clean and then back up a shared directory. Script execution requires users to have *root* privileges, so *root* user sets the SUID bit. An authorized user can now interrupt the script and gain a prompt as *root*.

Use **find** to locate SUID/SGID files and ensure no scripts have SUID or SGID bits set. Use **-exec** in combination with **find** to perform an action, such as a long listing on the results of the **find** command as shown below.

```
[root@CentOS6_A ~]# find /usr/sbin -perm -4000 -exec ls –l  {} \;
[root@CentOS6_A ~]# find /usr/sbin -perm -2000 -exec ls -l {} \;
[root@CentOS6_A ~]# find /usr/sbin -perm -6000 -exec ls -l {} \;

[root@CentOS6_A ~]# find /usr/sbin -perm -u=s -exec ls -l {} \;
[root@CentOS6_A ~]# find /usr/sbin -perm -g=s -exec ls -l {} \;
[root@CentOS6_A ~]# find /usr/sbin -perm -ug=s -exec ls -l {} \;
```

When searching for a file using –**perm** with the numeric method, removing the '- ' before the value searches for the explicit value. For example **-perm 0644** finds that exact permission, whereas –**perm -4000** finds any files with the SUID bit set (`rwsrw-r--`) and **-perm -2000** finds any files with the SGID bit set (`rwxrwsr--`).

Verifying SUID/SGID permissions on scripts is vital, but other system settings should be adhered to and included in a catalog of checklists as well. This creates a security baseline that can help reduce vulnerabilities. Address deviations from a baseline immediately and automate tasks for checklist compliance.

The Defense Information Systems Agency (DISA) maintains the security posture of the Department of Defense (DoD). DISA develops and uses Security Technical Implementation Guides (STIGs) to standardize security protocols within networks, servers, computers, and logical designs to enhance overall security. The STIG for SELinux OS is one example of a secure baseline. Other additions to a baseline may include software updates, security patches, system and software integrity checks, permissions, accounts, and access controls. Baseline settings are addressed throughout this module.

For additional security, use the **find** command to obtain a listing of files that have been made world-writable through incorrect permission assignment. A best practice is to remove user *other* write access to real files. If not removed, any user on the system can modify data in world-writable files. Locate these files and determine if permissions should be changed.

The **find** command produces many file types that may be omitted from the entire search, such as symbolic links (`l`), block devices (b), and character devices (c). Use the **!** (logical NOT) expression along with the **-type** option and file type, to prevent displaying undesired file types. Add any of the following options to the above command before the **-exec** to exclude these types of files from the search.

```
! -type l          ! -type c          ! -type b
```

Of course, omitting the **!** indicates that a particular file type should be included. The most common searches are for ordinary files (`f`). The example below searches from the beginning of the file system for ordinary files, with SUID permissions, and prints a long listing to the screen.

```
[root@CentOS6_A ~]# find / -type f -perm -4000 -exec ls -l {} \;
```

The example below searches the */home* directory for any files where user *other* has write permissions to data in a file and prints a long listing to the screen.

```
[root@CentOS6_A ~]# find /home -type f -perm -2 -exec ls -l {} \;
```

Note: *-perm -2* is the same value as *-perm -0002*.

The **!** can also be used with other options. The example below searches the */selinux* directory for world-writable files, whose file names do NOT begin with a `c'.

```
[root@CentOS6_A ~]# find /selinux -type f -perm -2 ! -name c*
```

Another common use of the **find** command is to locate empty files with the **-size** option. The example below locates all empty files located in the */home* directory, and sends a long listing of the files to */tmp/zero.list*. From there, it may be decided whether to keep or delete the empty file.

```
[root@CentOS6_A ~]# find /home -size 0 -exec ls -l {} \; > /tmp/zero.list
```

Complete Exercise 8-1 in Student Workbook

Permissions

Complete Exercise 8-2 in Student Workbook

Linux Command Refresher

Objectives

Day 2

At the end of this training day, students will:

- ❖ Recall the Linux environmental structure.
- ❖ Describe user components of the Linux system architecture.
- ❖ Describe kernel components of the Linux system architecture.
- ❖ Define phases of the Linux boot process and modify related initialization files.
- ❖ Identify and secure Linux system services at various run-levels.

Exercises

This training day includes the following exercises:

- ❖ Exercise 8-3, Configuring System Run-levels

JCAC-eTC

Complete self-study activities and assignments in the online training environment.

3 The Linux Architecture

Notable for its size and complexity while still being portable, Linux can be compiled to run on many processors and platforms with various architectures. The Linux OS is often referred to as Linux or GNU/Linux; however, Linux is actually the kernel of the OS and the overall central component. The application that makes the OS work is the GNU software, consisting of a Graphical User Interface (GUI), compilers, shells, tools, and editors, which are all user components.

Linux features two rings of protection to help prevent inadvertent programming errors and malicious activity that could harm processes or corrupt data structures. The kernel runs in the most privileged ring, known as **kernel mode**. The user applications execute in the least privileged ring, known as **user mode.**

Figure 1 Linux system architecture displays the architectural layers of a typical Linux OS. The layers encompass user mode, kernel mode, and hardware.

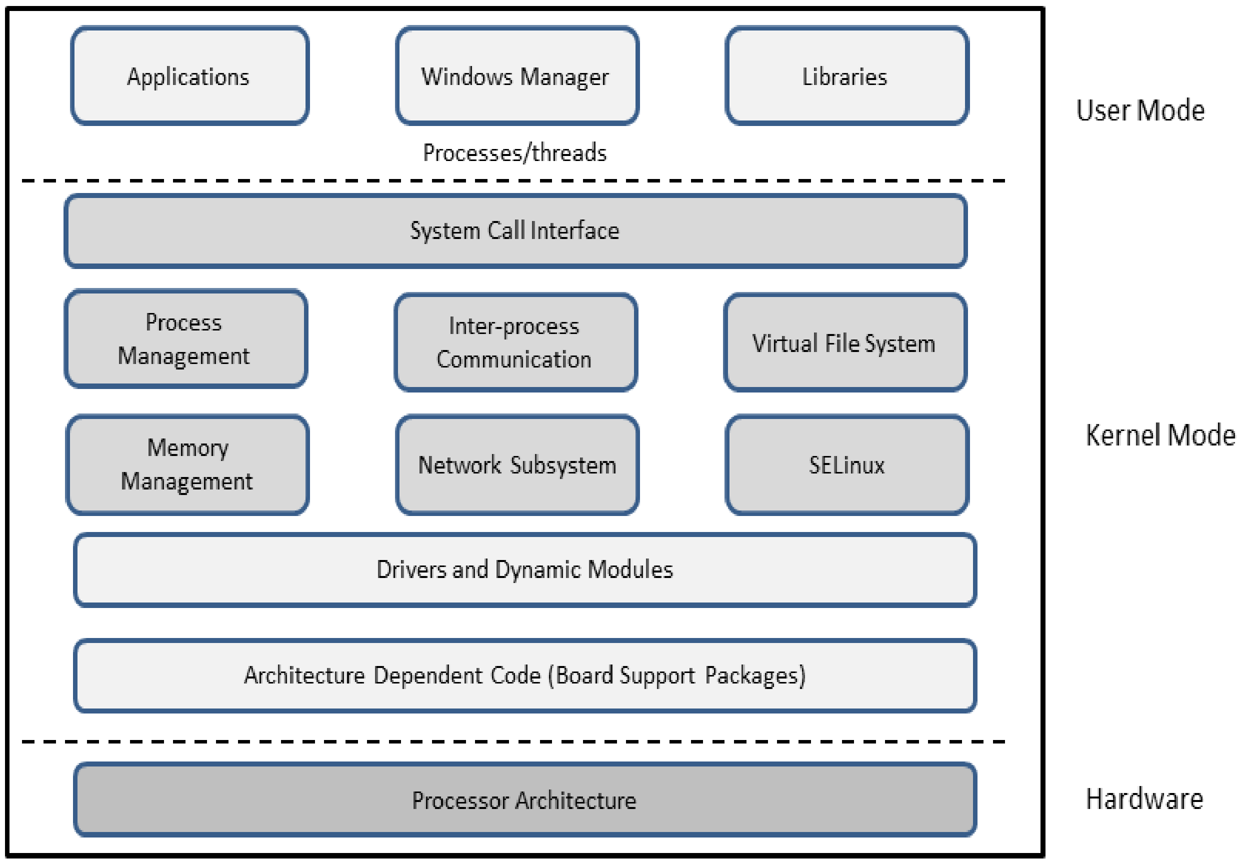


Figure 1. Linux system architecture.

3.1 User Mode Components

User mode includes processes/threads, applications, a windows manager, libraries, and a variety of other components; each having unique operating requirements and actions.

3.1.1 Processes and Threads

Running processes and threads request kernel services. User processes are protected from other user processes because each run in their own virtual address space. Every process is guaranteed 4GB virtual address space. A 32-bit Linux system reserves the upper region 1 GB of memory for the kernel and the remaining 3GB for the user.

3.1.2 Windows Manager

A windows manager defines how the GUI environment looks and how the system is accessed by users. The GUI evolved into complete desktop environments simplifying file management, system configuration, and computer management. Linux often uses MATE or GNOME windows managers. User applications and third-party programs that enhance a user's environment are located in */opt*.

3.1.3 Libraries (GNU C)

The GNU C Library (glibc) is an implementation of the standard C library used by many OSs and programs. Library functions act as interfaces for the abstraction of system calls. For example, *fork* and *execve* are functions in glibc and call the lower-level *fork()* and *execve()* system calls.

3.1.4 Applications

Applications are computer programs that perform certain tasks, functions, or activities that originate in user mode. When the processor receives a *syscall* instruction from an application, the instruction causes an exception, and control is transferred to an exception handler. Once a user process executes a system call and begins executing kernel code, the process can then operate in kernel mode. Due to the protection provided by this sort of isolation, crashes in user mode are rare and easily recoverable. That is why most of the code running on a computer executes in user mode.

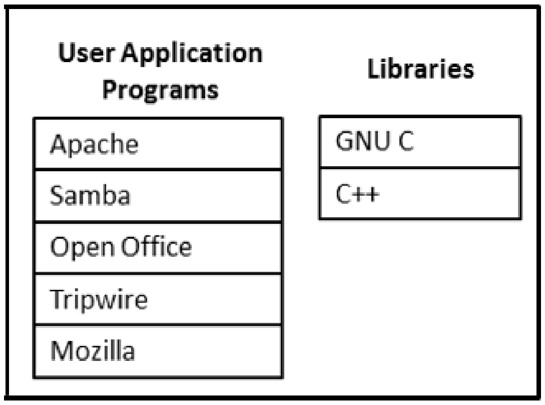


Figure 2. Examples of user applications and libraries.

Figure 3 displays how a user application reference a library in the Linux architecture.

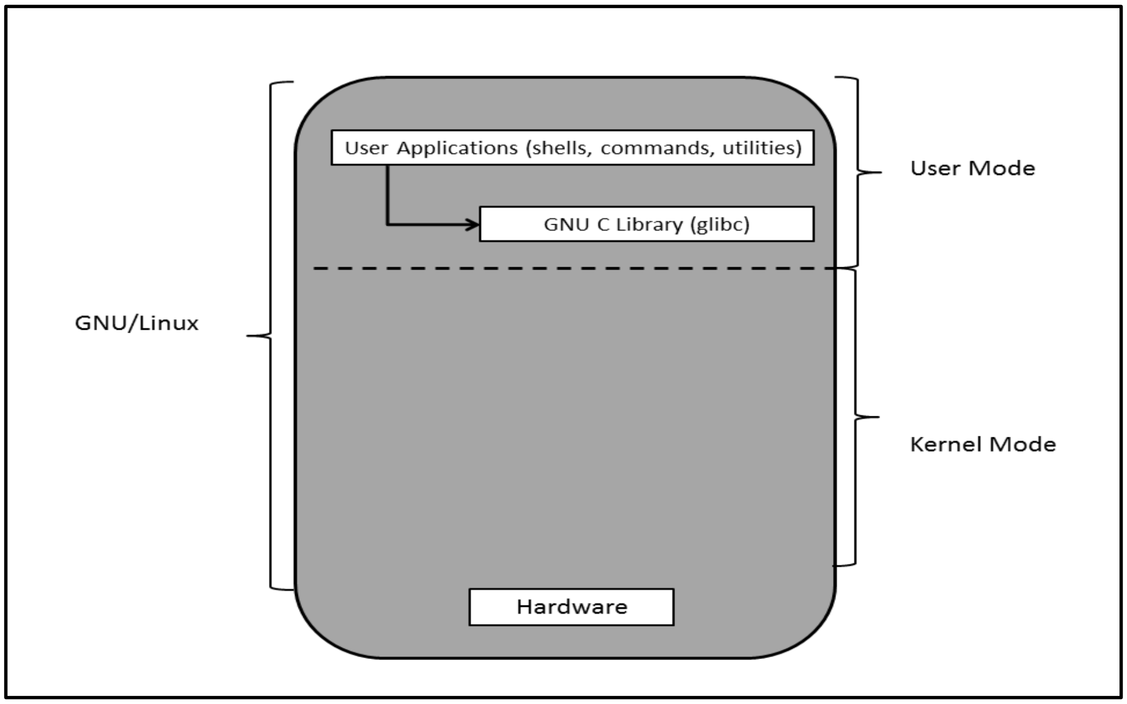


Figure 3. Linux user mode components.

3.1.5 Shells, Commands, and Utility Programs

In addition to applications and libraries, Table 2 displays other components that fall into the user mode category:

- Shells
- Commands
- Utility Programs

Table 2. User initiated processes

Shells	**Commands**	**Utility Programs**
sh	ls	tar
bash	mkdir	tcpdump
csh	cp	vi
tcsh	rm	fdisk
zsh	pwd	iptables

The shell, commands, and utilities are separate programs that are part of an OS distribution, but not part of the kernel, and are categories of applications programmed to use an API to request kernel services. APIs are implemented as system calls or via other means, including other APIs. The most common API for UNIX based systems is the Portable Operating System Interface for UNIX (POSIX)-defined API and its system calls because they closely resemble those of the C library.

A shell presents each user with a prompt, executes user commands, and supports a custom user environment. A shell is an interpreter and a scripting language with a specific set of functions. The **chsh** command retrieves information from a listing of available shells in */etc/shells*. Use `chsh -l` to list the available shells on a system.

Commands and utility programs provide system I/O. Utility programs run as a single command in support of computer usage such as file management, text editing, archiving, and maintenance. These are normally located in */sbin*.

3.2 Kernel Mode Components

Kernel mode is a privileged system area that resides between user mode and the hardware platform. In kernel mode, executing code has full, unrestricted access to hardware and can execute any CPU instruction and reference any memory address. This privileged area manages system resources and performs system services. The kernel is loaded into memory at system boot and is the core of the OS. Kernel memory is normally reserved for the lowest level, most trusted function of the OS to prevent kernel crashes. Kernel crashes are catastrophic and halt the entire PC. This type of crash is referred to as a kernel panic.

The kernel performs a privileged task on behalf of a calling process with the request coming via the API and a system call. Tasks may include reading or writing a file, sending commands to a device, creating a process, or managing memory. Kernel mode contains additional subsystems and other services linked in as loadable modules. Figure 4 displays some of the Linux architecture kernel mode components.

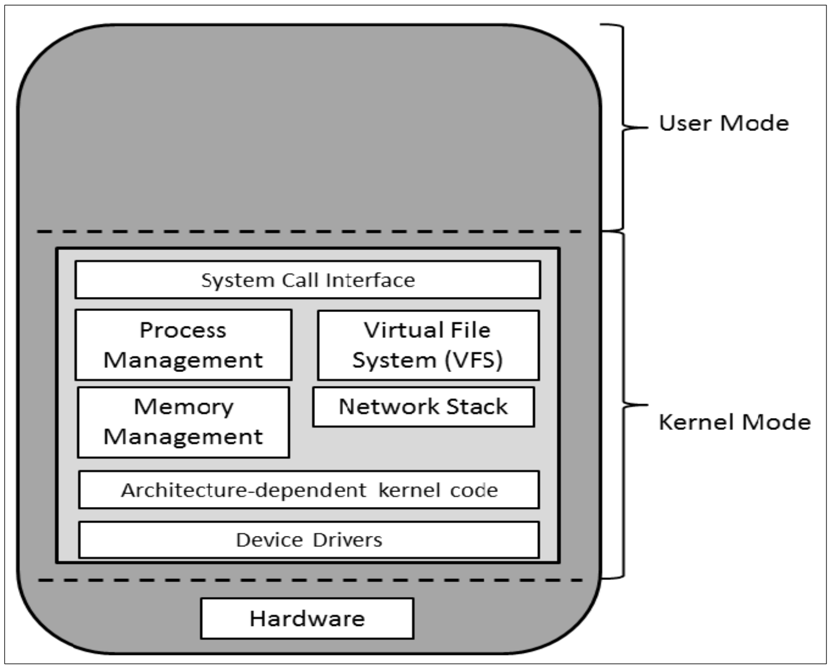


Figure 4. Linux kernel mode components.

3.2.1 System Call Interface (SCI)

The kernel communicates user requests to kernel services using the SCI. The SCI converts a process running in user mode to a protected kernel mode process. To implement a system call, control must be transferred through an architecture-specific feature such as a software interrupt or trap. Converting the system call to the requested kernel service allows a program to invoke protected kernel routines for performing system functions.

A system call table maintains a set of pointers to library functions that implement system calls. System calls are assigned a number that is passed to the kernel when requested. A trap transfers control so software can set up a register with the assigned system call number.

The CPU is placed into a privileged level, passing control to the kernel. The kernel decides if the program is allowed the request. If so, the kernel executes a set of instructions, returns the privilege level to that of the calling program, and then returns control to the calling program. If not allowed, the kernel returns an error to the calling program and the process is terminated or placed back in queue.

In user mode, the library function instructions are transferred to the kernel entry point, system_call(). If the system_call() function is valid, it is invoked. The system call is executed and transfers to the actual system call code.
Figure 5 illustrates this process.

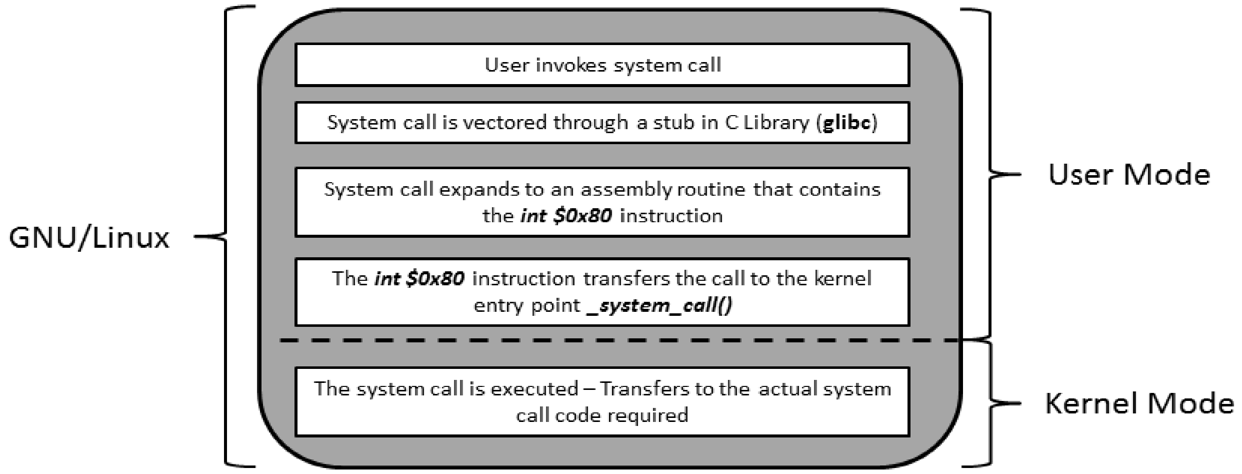

User invokes system call
System call is vectored through a stub in C Library (**glibc**)
System call expands to an assembly routine that contains the ***int $0x80*** instruction
The ***int $0x80*** instruction transfers the call to the kernel entry point ***_system_call()***
The system call is executed – Transfers to the actual system call code required
GNU/Linux
User Mode
Kernel Mode

Figure 5. System call flow.

The library's wrapper function name is often similar to the system call invoked. The wrapper places all arguments to be passed to the system call in the proper memory location and sets a unique system call number for the kernel to call. The kernel begins executing code for system calls on behalf of the user. After the library function executes, the call stack is unwound to restore the processor state. Control is returned to the user program. Library functions execute predominantly in user mode unless invoking a system-level service request; part of the standard C library and are either static (.a) or a dynamically linked share object (.so)

3.2.2 Process Management

Process management ensures the proper and timely execution of processes. Each process includes a set of resources such as open files, pending signals, internal kernel data, processor state, address space, one or more threads of execution, and a data section called the stack.

A Lightweight Process (LWP) is a process that runs in user mode on top of a single kernel thread and shares its address space and system resources with other LWPs within the same process. Every process consists of at least one thread. All threads are considered LWPs and are objects of activity within a process. An LWP includes a unique program counter, process stack, and a set of process registers. The kernel uses a scheduler to share CPU time between LWPs. This ensures each process gets time on the CPU with minimal conflicts.

Operating Systems	The kernel's schedule dispatcher uses scheduling algorithms (preemptive and non-preemptive) to determine which process to run when the CPU becomes available.

See Information Sheet 8-1 in Student Workbook

Process Lifecycle Flowchart

3.2.3 Memory Management

The kernel is responsible for managing memory resources using pages, normally 4KB chunks. The memory management scheme keeps track of what pages are full, partially full, or empty, and adjusts dynamically in size based on system needs. When memory is exhausted, pages are moved onto disk or swapped.

3.2.4 Virtual File System (VFS)

The VFS component implements file and file system-related interfaces provided to user mode programs. File systems (e.g., EXT3, EXT4, NTFS, UFS, and XFS), rely on a VFS to coexist and inter-operate. The **blkid** command displays the file system type. VFS enables programs to use standard UNIX/Linux system calls to read and write to different file systems, even on different media.

```
[root@CentOS6_A ~]# blkid
/dev/sda1: UUID=”3ddf6cb3-5a38-46f9-8b0e-98ecbe7d795a” TYPE=”ext4”
```

Figure 6 displays the VFS in action. In this example, VFS copies data from one media type and file system to another media type running a different file system.

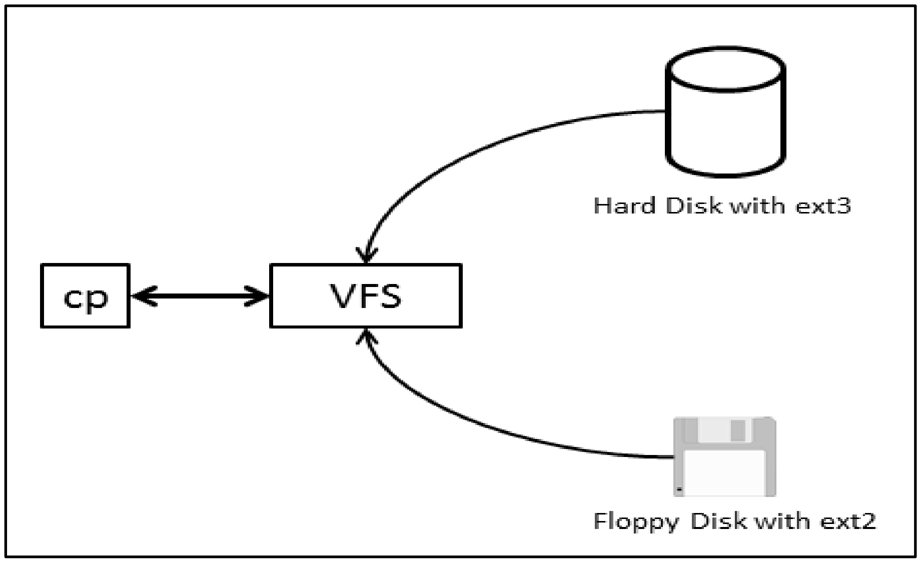


Figure 6. VFS in action.

3.2.5 Network Subsystem

The network subsystem component provides an API to resources that send data across a network. The following steps illustrate the process:

Step 1: Linux presents a send socket buffer to a user application by using a file descriptor.

Step 2: When a *write* system call is requested, user mode data is copied to kernel memory and then appended to the socket buffer to ensure data is sent in the proper order.

Step 3: The device driver communicates with the NIC communications protocol for instructions.

3.2.6 Architecture-dependent Kernel Code

Architecture-dependent code, or Board Support Package (BSP), refers to elements of the Linux kernel used by a particular architecture for normal operation and efficiency. BSP is essential code that allows a computer hardware device to work with the computer OS. The BSP contains a small program called a boot loader or boot manager that places the OS and device drivers into memory. The */usr/src/kernels/2.6.32-279.el6.x86_64/arch* subdirectory defines this portion of the kernel source. These directories and files collectively form the BSP. Each architecture subdirectory contains other subdirectories that focus on different aspects of the kernel, such as boot, kernel, memory management, power, and video.

3.2.7 Device Drivers

Nearly all device control operations are performed by kernel code that is specific to a device. The kernel uses files in */dev* to make device services available to user programs. A device driver is a low-level program that allows the kernel to communicate with a specific piece of hardware. The driver performs as an interpreter for the device.

Two types of device drivers exist: those permanently coded into the kernel when it is built, and modular kernel drivers that are loaded, unloaded, or reconfigured without a system restart. Distributors want to support all cards and devices possible with Linux. However, if these were compiled into the kernel, its size would negate its portability. In modular systems, manufacturers can provide a stripped-down kernel and a comprehensive set of device drivers.

3.3 Hardware Components

Identifying installed hardware components is an essential element of evaluating computer architectures in support of security requirements and is a vital part of system enumeration. The system's hardware components and other relevant system information are maintained in the computer's Desktop Management Interface (DMI) table.

`dmidecode`	Dumps the contents of the DMI table to the screen in human-readable format.		
	Options:	`-s` (string)	Only display the value of the DMI string identified by a keyword
		`-t` (type)	Type can be either a DMI type number, or a comma-separated list of type numbers, or a keyword

Syntax:

```
dmidecode -s processor-version
dmidecode -t memory
```

4 Linux Boot Process

When powered on, a computer goes through a sequence of phases ensuring the OS is properly running and ready to accept logins. Boot phases vary depending on OS, platform, firmware, and system initialization methods.

CentOS version 6 (CentOS6) uses a modified System V (SysV) initialization manager called Upstart. This Linux topic considers CentOS 6 as SysV (service and chkconfig commands). With Upstart or SysV, the init process brings the system to a particular run-level. Run-levels dictate the startup environment of the system (i.e., what services are running). Since init is the first process started by the kernel, if it does not start, no other processes start. This creates a kernel panic.

CentOS version 7 (CentOS7) uses a more secure, but less controllable, system initialization manager called systemd. Like init, the systemd process is the first process to start. Systemd enhances boot time by starting multiple processes in parallel vice sequentially. Additional comparisons of init and systemd appear later in the topic.

Familiarization with the boot process is the first step in troubleshooting when a system does not boot or is not responding as intended. Analysts use the boot phases to determine firmware and kernel versions, modify default run-level states, and gain root user access when the boot loader executes.

Figure 7 identifies the boot phases of CentOS using the SysV (*init)* initialization manager. This example of the boot phase uses a Basic Input/Output System (BIOS) boot partition with Master Boot Record (MBR) executing Grand Unified Bootloader (GRUB).

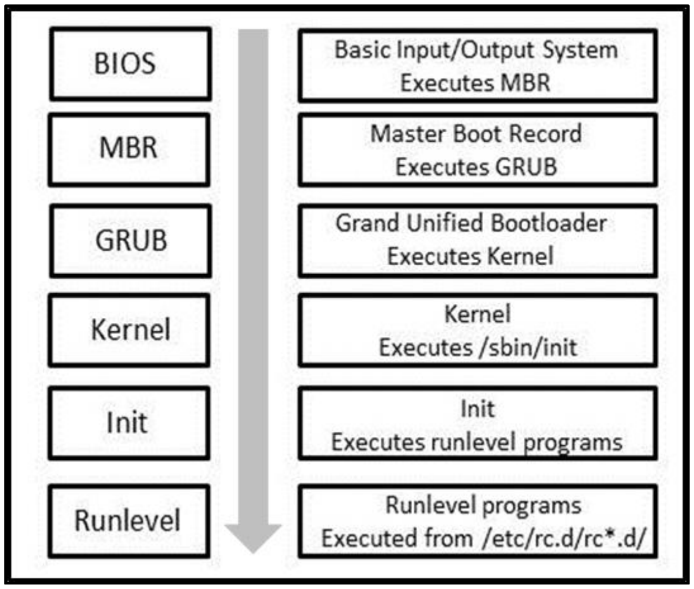

BIOS
MBR
GRUB
Kernel
Init
Runlevel
Basic Input/Output System Executes MBR
Master Boot Record Executes GRUB
Grand Unified Bootloader Executes Kernel
Kernel Executes /sbin/init
Init Executes runlevel programs
Runlevel programs Executed from /etc/rc.d/rc*.d/

Figure 7. Linux boot process.

4.1 BIOS Phase

After applying system power, BIOS performs a Power On Self Test (POST) and executes the MBR.

- Performs system integrity checks
- Builds a device tree (locates floppies, hard drives, CD-ROMs)
- Executes MBR

Unified Extensible Firmware Interface (UEFI) partitioning is available and supports MBR and GUID Partition Table (GPT). The *efibootmgr* must allow addition of the GRUB EFI boot entry. UEFI firmware is not implemented consistently by all manufacturers.

4.2 MBR Phase

The MBR located at the beginning of a bootable disk contains the primary bootloader, partition table, and MBR validity check. Bootloaders vary in name and function. Types include GNU GRUB, LILO, BURG (derived from GRUB), and syslinux which is an assortment of lightweight boot loaders that support many file systems. GRUB is used in the example that follows.

- MBR executes the bootloader (GRUB)

4.3 GRUB Phase

GRUB is dynamically configurable with the capability to make changes during boot. Bootloader changes include altering boot entries, selecting different kernels, and modifying the initial RAM disk (*initrd*). *Initrd* is an initial root file system with a limited set of directories and executables. Mounted prior to loading the root file system, *initrd* is bound to the kernel and loaded as part of the kernel boot process.

- If multiple OSs are available and a choice is not made on the OS selection screen, *init* loads the default kernel specified in the GRUB configuration file (*/boot/grub/grub.conf*).
- Loads and executes *kernel* and *initrd* images.

Displayed below is an abbreviated portion of this file.

```
[root@CentOS6_A ~]# less /boot/grub/grub.conf
# grub.conf generated by anaconda
…
title CentOS (2.6.32-279.el6.x86_64)
         root  (hd0,0)
         kernel  /vmlinuz-2.6.32-279.el6.x86_64 ro root=UUID=5b643934….
(truncated)…rhgb quiet
         initrd  /initramfs-2.6.32-279.el6.x86_64.img
[root@CentOS6_A ~]#
```

Obtaining the kernel version information is a useful fingerprinting tool. The **`uname -a`** command reveals the kernel name (Linux), version (2.6.32), release (279.el6), and the CPU architecture (x86_64) information. On a CentOS 7 system, the **`hostnamectl`** command reveals detailed information.

Editing the GRUB menu and modifying the kernel entry as the system boots, may allow *root* user access if the system is not secure. Older Linux versions enable this feature. SELinux (CentOS 7) removed this ability, making it more difficult to obtain *root* access. CentOS6, used in these VM's has this capability, but perfoming the process is beyond the scope of this topic.

A first step in Defensive Cyberspace Operations (DCO) is reducing unauthorized system access by implementing boot security. Modifying the GRUB menu to require authentication is simple, but rarely implemented. The following creates a hashed password for the GRUB menu.

- Create an MD5 hashed password:

```
[root@CentOS6_A ~]#  /sbin/grub-md5-crypt
Password: abcd1234
Retype password: abcd1234
$1$qhqh.1$7MQxS6GHg411OFMdnDx95.
```

- Edit */boot/grub/grub.conf* and add the password entry following the timeout entry

```
timeout=5
password --md5 <pwhash from grub-md5-crypt command above>
```

4.4 Kernel Phase

During the kernel phase, the Linux kernel sets up the rest of the system by mounting the root file system and executing */sbin/init*.

- Mounts root file system specified by the *root=* entry in *grub.conf* file
- Executes */sbin/init* daemon, the very first process started with a process ID (PID) of 1

4.5 Init Phase

Since the release of SysV, the init process (daemon) placed and maintained the system in a specified run-level or state. At any point in time, the system is in one of up to eight possible run-levels. The number of run-levels varies between Linux versions. The initialization phase of system boot invokes the init process. init searches the initialization table (*/etc/inittab*) for the default run-level, identified by the initdefault entry. If the entry exists, all processes and scripts relating to that run-level execute.

The *inittab* file contains the system default line: "*id:5:initdefault:*". This line specifies what run-level starts at boot. The default run-level in Linux is 5. If an initdefault entry does not exist, the system boots to a terminal login prompt. Each field is separated by a colon. The first field is simply an identifier <*id*>. The second field of each line identifies the <*run level*> for which the specified <*action*> (third field) should be taken. A fourth field specifies a process to execute. Most lines in *inittab* are informational comments. *inittab* is a script read in linear fashion from top to bottom. The following command displays the */etc/inittab* on CentOS6_A. A # symbol at the beginning of a line indicates a comment field.

The only line init reads and executes in the example below is the initdefault line.

```
[root@CentOS6_A  ~]# more /etc/inittab
# inittab is only used by upstart for the default runlevel.
# ADDING OTHER CONFIGURATION HERE WILL HAVE NO EFFFECT ON YOUR SYSTEM.
# System initialization is started by /etc/init/rcS.conf
# Individual runlevels are started by /etc/init/rc.conf
# Ctrl-Alt-Delete is handled by /etc/init/control-alt-delete.conf
# Terminal gettys are handled by /etc/init/tty.conf and
# /etc/init/serial.conf, with configuration in /etc/sysconfig/init.
# For information on how to write upstart event handlers, or how
# upstart works, see init(5), init(8), and initctl(8).
# Default runlevel.  The runlevels used are:
#   0 - halt (Do NOT set initdefault to this)
#   1 - Single user mode
#   2 - Multiuser, without NFS (The same as 3, if you do not have networking)
#   3 - Full multiuser mode
#   4 - unused
#   5 - X11
#   6 - reboot (Do NOT set initdefault to this)
#
id:5:initdefault:
[root@CentOS6_A  ~]#
```

4.6 Run-level Phase

Init executes the associated run control (rc) scripts located in one or more *rc* directories. Linux supports the following run levels:

Normal Run-levels for Linux:

0	Halt mode	All processes terminated; orderly halt (Do not make initdefault).
1	Single-user mode	Used to perform administrative tasks to this system.
2	Multi-user mode	Allows users to access the system; limited network services.
3	Full multi-use mode	Same as multi-user, but includes network services.
4	User-defined mode	Not specified by system. Universally defined and customizable.
5	X11	Default; multi-user, network services and X-windows display manager (X11).
6	Reboot mode	All processes terminated; system is gracefully rebooted.

A directory for each run-level is located in */etc/rc.d* directory. For example, run-level 0 has a directory called */etc/rc.d/rc0.d/*, that contains a symbolic link to the associated rc script located in */etc/init.d*.

- Run-level 0 */etc/rc.d/rc0.d/*
- Run-level 5 */etc/rc.d/rc5.d/*

Linux also provides a link to each run-level directory, in the */etc* directory. For instance, */etc/rc5.d* is a symbolic link to */etc/rc.d/rc5.d*. For ease of use, */etc/rc*.d* is used in this module.

Functions performed at different run-levels include:

- Perform system maintenance tasks
- Check and mount file systems
- Start various processes
- Spawn a login (getty or mingetty process)

The following commands are associated with run-levels.

`init`	Change current run-level. i.e., **`init 6`**
`who –r`	Used to determine current run-level status.

4.7 Run Control (rc) Scripts

The rc scripts control the automatic boot process called by *init*. Run-levels are modified by *root* account and may require multiple rc scripts. For example, run-level 5 executes each script in */etc/rc.d/rc5.d* directory. Run-level 5 is the Linux default.

Master copies of all rc scripts are located in the */etc/init.d* directory. The *init.d* directory contains initialization and termination scripts for changing initialization states.

The rc directories contain scripts that bring the system to a particular run-level. Script names begin with a 'K' (kill) or an 'S' (spawn). Scripts beginning with a 'K' execute first and kill associated processes or services. Afterwards, scripts beginning with an 'S' execute and spawn (start) associated processes or services. The numerical value following the 'K' or 'S' is used for sequencing. Scripts execute in order of dependencies. For example, network services must start prior to running the Apache web service.

The following displays the CentOS default run-level (5) directory where most service scripts start. All scripts in the rc directories are linked to the corresponding entry in /etc/rc.d/init.d.

```
[root@CentOS6_A rc5.d]# ls
K01smartd          K99rngd            S24rpcgssd
K05wdaemon         S01sysstat         S25cups
K10psacct          S02lvm2-monitor    S25netfs
K10saslauthd       S03vmware-tools    S26acpid
K15httpd           S08ip6tables       S26haldaemon
K35nmb             S08iptables        S26udev-post
<output truncated>
K75ntpdate         S20kdump           S82abrt-oops
K75quota_nld       S22messagebus      S90crond
K84wpa_supplicant  S23NetworkManager  S95atd
K87restorecond     S24avahi-daemon    S99firstboot
K89rdisc           S24nfslock         S99local
[root@CentOS6_A rc5.d]#
```

The following displays scripts run during a shutdown (0). Notice in run-level 0 (Halt mode), all scripts are killed and then **S00killall** and **S01halt** are spawned. This is a graceful shutdown.

```
[root@CentOS6_A rc0.d]# ls
K01smartd                  K50snmptrapd       K87irqbalance
K02avahi-daemon            K50xinetd          K87restorecond
K05atd                     K60crond           K87rpcbind
K05wdaemon                 K60nfs             K88auditd
K10cups                    K69rpcsvcgssd      K88rsyslog
K10psacct                  K74acpid           K89portreserve
K10saslauthd               K74haldaemon       K89rdisc
K15httpd                   K74ntpd            K90network
K16abrt-ccpp               K75netfs           K92ip6tables
K16abrtd                   K75ntpdate         K92iptables
K16abrt-oops               K75quota_nld       K95firstboot
K25sshd                    K75udev-post       K99cpuspeed
K30postfix                 K80kdump           K99lvm2-monitor
K30spice-vdagentd          K83bluetooth       K99rngd
K35nmb                     K83nfslock         K99sysstat
K35smb                     K83rpcgssd         K99vmware-tools
K43vmware-tools-thinprint  K84NetworkManager  S00killall
K50dnsmasq                 K84wpa_supplicant  S01halt
K50netconsole              K85mdmonitor
K50snmpd                   K85messagebus
[root@CentOS6_A rc0.d]#
```

4.8 Securing Startup Scripts (SysV)

Starting all services at boot time makes a system vulnerable. Scripts start or stop services at boot for different run-levels. Service status is changed by renaming the script to prevent it from running at boot, using the **chkconfig** command, using the **service** command, or running the script from */etc/init.d*. Changes update the rc script name and update the text file in */etc/xinetd.d/<service>* directory.

Because Linux is case sensitive, renaming a script changes how it is treated. For instance, *S50bluetooth* starts by default in run-level 5. To prevent the service from starting at boot, rename the script to *s50bluetooth.* The *init* daemon only recognizes scripts in the rc directories that begin with an uppercase ‘K’ or ‘S’, therefore *s50bluetooth* is ignored at boot time.

```
   [root@CentOS6_A ~]# mv /etc/rc5.d/S50bluetooth /etc/rc5.d/s50bluetooth
```

Instead of renaming the script, use the **chkconfig** command. The command updates and queries run-level information for system services running from */etc/rc.d/init.d*. **chkconfig** does not specifically start or stop a service, it only adds or removes the service to those daemons that run at boot. The -- *list* option lists all services that **chkconfig** knows about and whether they are stopped or started in each run-level. It uses an on/off flag to indicate if the service starts when the system switches to that run-level.

The following example displays a partial list of available services. The service name is located in the first column and the remaining columns indicate if the service starts at the referenced run-level (0-6). At the bottom of the list are the network-based services that fall into the *xinetd* category. The 'x' means extended. These services are located in */etc/xinetd.d* directory. Using **chkconfig** to change network-based services is immediate and requires no reboot. Changes made to non-networking services do not take effect until switching to that run-level or through a system reboot.

```
[root@CentOS6_A ~]# chkconfig --list
NetworkManager 0:off   1:off   2:on   3:on   4:on   5:on   6:off
abrt-ccpp      0:off   1:off   2:off  3:on   4:off  5:on   6:off
abrt-oops      0:off   1:off   2:off  3:on   4:off  5:on   6:off
<output truncated>
wpa_supplicant 0:off   1:off   2:off  3:off  4:off  5:off  6:off
xinetd         0:off   1:off   2:off  3:on   4:off  5:on   6:off

xinetd based services:
      chargen-dgram:     off
      chargen-stream:    off
      daytime-dgram:     off
      telnet:            on
      time-dgram:        off
      time-stream        off
[root@CentOS6_A ~]#
```

The following example turns the SSH service off for all run-levels and sets the http service on for levels 4 and 5:

```
[root@CentOS6_A ~]# chkconfig sshd off
[root@CentOS6_A ~]# chkconfig --level 45 httpd on
```

The **service** command starts, stops, or restarts many services located in */etc/init.d*. Changes are immediate, but the **service** command does not update the rc script name. Some scripts may support *restart* and *status* options, but all support at least a *stop* and *start.*

Services can also be directly controlled in the */etc/init.d* directory without the use of **chkconfig** or **service** commands. As shown below, both commands perform the same function.

```
[root@CentOS6_A ~]# /etc/init.d/smb {start / stop / restart / status}
[root@CentOS6_A ~]# service smb {start / stop / restart / status}
```

Examples of securing services that use rc scripts are listed below.

Disable ip6tables:

Disable ip6tables if not using IPv6. Either of the following two commands makes changes to the current session only, with immediate effects. Changes revert after a reboot.

```
[root@CentOS6_A ~]# /etc/init.d/ip6tables {start / stop}
[root@CentOS6_A ~]# service ip6tables {start / stop / status}
```

Either of the following two commands makes changes only after a reboot.

```
[root@CentOS6_A ~]# mv /etc/rc5.d/S08ip6tables /etc/rc5.d/s08ip6tables
[root@CentOS6_A ~]# chkconfig ip6tables {on / off}
```

Disable Apache:

Disable Apache when the machine is not running Apache web services. Either of the following two commands makes changes to the current session only, with immediate effects. Changes revert after a reboot.

```
[root@CentOS6_A ~]# /etc/init.d/httpd {start / stop}
[root@CentOS6_A ~]# service httpd {start / stop / status}
```

Either of the following two commands make changes only after a reboot.

```
[root@CentOS6_A ~]# mv /etc/rc5.d/S15httpd /etc/rc5.d/s15httpd
[root@CentOS6_A ~]# chkconfig httpd {on / off}
```

Disable Telnet:

The Telnet service is used to interact with another host via a CLI. The *xinetd* controls its startup and is not normally started with an rc script. It is covered in detail in the network section of this module. Changes are immediate and permanent.

Having the Telnet service running poses a security risk, therefore it is rarely used today. One of the biggest disadvantages of this protocol is that all data, including names and passwords are sent in clear text. Telnet should be replaced by a more secure protocol such as SSH and the entire suite of SSH tools. Server and client machines with the Telnet service on are a deviation from normal, secure operations.

```
[root@CentOS6_A ~]# chkconfig telnet {on / off}
[root@CentOS6_A ~]# vi /etc/xinetd.d/telnet {disable = yes}
```

4.9 Securing Startup Scripts (Systemd)

CentOS 7 uses *systemd* to enable, start, stop, or disable services. **`Service`** and **`chkconfig`** commands are available for backwards compatibility, but the preferred replacement for these is **`systemctl`**. Using **`Service`** or **`chkconfig`** on CentOS7 results in a note after the command is executed saying: "Redirecting to /bin/systemctl …". Listed below are key **`systemctl`** options:

• enable	Automatically starts at boot, similar to **`chkconfig <name> on`** in SysV.
• start	Service immediately starts, similar to **`service <name> start`** in SysV.
• disable	No longer starts on boot.
• stop	Stops the service, but starts again at next reboot.
• static	Dependent on other services, controlled automatically

The following examples list all available services and their current configurations.

```
SysV      [root@CentOS6_A ~]# chkconfig --list
Systemd   [root@CentOS7A ~]# systemctl list-unit-files --type=service
```

The following examples determine if a particular service is set to start at boot (enabled).

```
SysV      [root@CentOS6_A ~]# chkconfig --list | grep httpd
Systemd   [root@CentOS7A ~]# systemctl <status or is-enabled> httpd
```

The following examples enable a particular service (httpd) to start at boot.

```
SysV      [root@CentOS6_A ~]# chkconfig httpd on
Systemd   [root@CentOS7A ~]# systemctl enable httpd
```

The following examples compare restarting *sshd* with **service** and then with **systemctl**.

```
SysV      [root@CentOS6_A ~]# service sshd restart
Systemd   [root@CentOS7A ~]# systemctl restart sshd
```

The following examples compare stopping *httpd* with **service** and then with **systemctl**.

```
SysV      [root@CentOS6A ~]# service httpd stop
Systemd   [root@CentOS7A ~]# systemctl stop httpd
```

The following examples compare starting *httpd* with **service** and then with **systemctl**.

```
SysV      [root@CentOS6_A ~]# service httpd start
Systemd   [root@CentOS7A ~]# systemctl start httpd
```

CentOS6 uses **iptables** and CentOS7 uses **firewalld** to determine the firewall's status. The status option can be replaced with stop and start for an immediate action.

```
SysV       [root@CentOS6_A ~]# service iptables status
Systemd    [root@CentOS7A ~]# systemctl status firewalld
```

A service must be enabled before it is started. A key difference in the commands are operation order. SysV – Command, Service, argument. Systemd – Command, argument, Service.

Complete Exercise 8-3 in Student Workbook

Configuring System Run-levels

Objectives

Day 3

At the end of this training day, students will:

- ❖ Utilize Linux utilities to view system calls used during command execution.
- ❖ View and manipulate Linux tunable kernel module parameters.
- ❖ View and set limits on files written by the shell and its child processes.
- ❖ Recognize and demonstrate process level interrupts, interrupt requests, and signals.
- ❖ Recognize and differentiate the functions of file systems and their components.
- ❖ Create hard links and symbolic links, and understand their necessity.
- ❖ Identify differences in file types and their locations.

Exercises

This training day includes the following exercises:

- ❖ Exercise 8-4, System Calls
- ❖ Exercise 8-5, Linux Kernel Tuning
- ❖ Exercise 8-6, Process Management
- ❖ Exercise 8-7, File and Directory Linking and File Type Recognition

JCAC-eTC

Complete self-study activities and assignments in the online training environment.

5 OS Internals

This section provides a comprehensive introduction to the Linux OS. It emphasizes the design of a Linux kernel to include system calls, loadable kernel modules (LKMs), and processes. These fundamental principles provide students with a solid understanding of the key structure and mechanics. With comprehension of this design, the effects of using system calls, the benefits of a modular OS, and how to view and control processes and their states becomes clear.

5.1 Types of System Calls

UNIX systems support various system calls used internally by native applications. They are basically functions used within the kernel. System calls can be grouped into categories: process control, file and device management, information management, communication, and protection. Section 2 (system calls) of the man pages further defines the system calls. To find information about a system call, such as *open*, execute the command `man 2 open`. Documentation may display parenthesis after a system call name to indicate arguments.

System calls for process control:

fork	Used to create a child process in the image of the parent process
wait	Blocks the calling process until its child process exits or signal is received
execve	Executes the program pointed to by filename, an executable or a script
exit	Terminates the current process and performs a cleanup afterwards
kill(sig, PID)	Sends a signal to another process to terminate it

System calls for file or device management (Input/Output):

creat	Creates and opens a file
open	Opens existing file/device
close	Closes all files associated with terminating process
read	Reads data from a file/device
write	Writes data to a file/device
ioctl	Input/output control of device-specific operations

System calls for information management:

getpid	Returns the PID of the calling process
uname	Returns system information pertaining to the platform

System calls for communication:

pipe	Creates a unidirectional data channel that can be used for Inter-Process Communication (IPC)
mmap	Creates a new mapping in the virtual address space of the calling process

System calls for protection:

umask	Sets the calling process' file mode creation mask (umask) to mask
chmod	Changes a file's mode bits

5.2 Examining System Calls

Most Linux systems have built-in commands to examine system calls accessed during command execution. Tracing a program provides information about a system and the system calls used. System calls and signals occur when the user interfaces with the kernel and vice versa. Examining the boundary helps isolate issues, provide sanity checks, and capture race conditions.

Linux uses the **`strace`** command to diagnose and debug programs when the source code is unavailable. Listed below are options and syntax supported by **`strace`** and used in this module.

`strace` Intercepts and records system calls that are called by a process.

Options:	**-c** <path>	Count time, calls, and errors for each system call
	-e *<expr>*	Traces a system call event (i.e., trace only "open") (read, write, open, …)
	-o *<filename>*	Each process trace is written to the filename given

Syntax:
`strace –c` <directory>/<filename>
`strace –e` <systemcall> <directory>/<filename>
`strace –o` <destination filename> <process or cmd>

Examples:

```
[root@CentOS6_A ~]# strace –c /bin/hostname
CentOS6_A
% time seconds       users/call     calls    errors  syscall
------  -------  ----------      -----    ------   -----------------
-nan    0.00000           0    3       read
-nan    0.00000           0    1       write
-nan    0.00000           0    5       open
-nan    0.00000           0    5       close
<Output truncated>
[root@CentOS6_A ~]#
```

```
[root@CentOS6_A ~]# strace –e open  /bin/hostname
open(*/etc/ls.so.cache", 0_RDONLY)  = 3
open(*/lib64/libselinux.so.1", 0_RDONLY)  = 3
open(*/lib64/libc.so.6", 0_RDONLY)  = 3
<Output truncated>
[root@CentOS6_A ~]#
```

```
[root@CentOS6_A ~]# strace –o /tmp/my.lstrace /bin/hostname
CentOS6_A
[root@CentOS6_A ~]#
```

The *execve()* system call line is described below:

execve("/bin/date", ["date"], [/* 34 vars */]) = 0

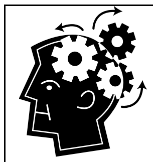
- *execve(),* system call name; invokes **date** program, which outputs date information
- 1st argument, "/bin/date" is the name of the program to run
- 2nd argument, ["date"] is a single element
- 3rd argument, [/* 34 vars */] represents the environmental list
- 0 is the return value

The example below shows **strace** executed on the date program.

```
[root@CentOS6_A /]# strace date
execve("/bin/date", ["date"], [/* 38 vars */]) = 0
brk(0)                                  = 0x10f1000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff805fae000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=53057, ...}) = 0
mmap(NULL, 53057, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ff805fa1000
close(3)                                = 0
open("/lib64/librt.so.1", O_RDONLY)     = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0@! \0273\0\0\0"..., 832) = 832

<output truncated>

write(1, "Tue Mar 22 10:54:58 PDT 2016\n", 29Tue Mar 22 10:54:58 PDT 2016 ) = 29
close(1)                                = 0
munmap(0x7ff805fad000, 4096)            = 0
close(2)                                = 0
exit_group(0)                           = ?
[root@CentOS6_A /]#
```



Complete Exercise 8-4 in Student Workbook

System Calls

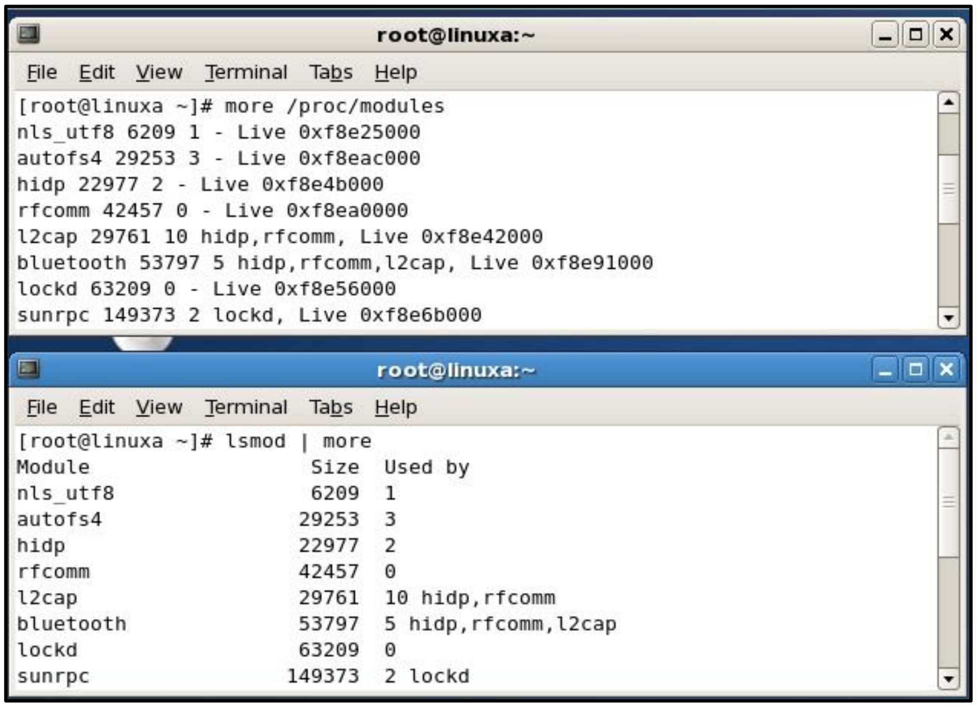
5.3 Loadable Kernel Modules (LKMs)

LKMs add or remove code from the Linux kernel during execution. This is ideal for device drivers. Devices require a driver file containing code the kernel uses to communicate with that device. Without LKMs, the kernel has to be recompiled each time new hardware is installed or a driver is updated, followed by a system restart. As well, code for every driver needed would exist in the kernel, resulting in a very large kernel.

LKMs are loaded at run time but do not execute in user space since they are a part of the kernel. LKMs run in kernel space and applications run in user space, each having unique memory address spaces that do not overlap. Kernel services are available to user space via system calls. The following commands are used to view, modify, and set limits on associated LKMs.

5.3.1 lsmod

The `lsmod` command lists currently loaded/active modules by nicely formatting the contents of */proc/modules*. Figure 8 compares the textual output with the results of executing `lsmod`.


root@linuxa:~
File Edit View Terminal Tabs Help

```
[root@linuxa ~]# more /proc/modules
nls_utf8 6209 1 - Live 0xf8e25000
autofs4 29253 3 - Live 0xf8eac000
hidp 22977 2 - Live 0xf8e4b000
rfcomm 42457 0 - Live 0xf8ea0000
l2cap 29761 10 hidp,rfcomm, Live 0xf8e42000
bluetooth 53797 5 hidp,rfcomm,l2cap, Live 0xf8e91000
lockd 63209 0 - Live 0xf8e56000
sunrpc 149373 2 lockd, Live 0xf8e6b000
```

root@linuxa:~
File Edit View Terminal Tabs Help

```
[root@linuxa ~]# lsmod | more
Module                  Size  Used by
nls_utf8                6209  1
autofs4                29253  3
hidp                   22977  2
rfcomm                 42457  0
l2cap                  29761  10 hidp,rfcomm
bluetooth              53797  5 hidp,rfcomm,l2cap
lockd                  63209  0
sunrpc                149373  2 lockd
```

Figure 8. lsmod command source.

5.3.2 sysctl

The Linux kernel is flexible and modified on the fly by dynamically changing or “tuning” some of its parameters. The */proc/sys* directory contains subdirectories for all tunable kernel module parameters and their values.

The **`sysctl`** command provides an interface that allows viewing and changing kernel parameters. A full list of kernel parameters and their current settings is viewed by running **`sysctl`** with the **`-a`** option. Changes or "writes" to kernel parameter values for the current session are performed using the **`-w`** option. These changes only affect the currently running kernel and are lost on reboot. **`sysctl`** also has an associated configuration file within */etc*, commonly named *sysctl.conf*. When the system boots and the **`sysctl`** command is executed using the **`-p`** option, kernel parameters and values within *sysctl.conf* are read. Changes are saved and remain persistent after a reboot.

Subdirectories in */proc/sys* identify which parameters are modifiable. A step in optimizing system performance is tuning the *file-max* parameter located in */proc/sys/fs*. This parameter limits the maximum number of files a user can have opened per login session. Use caution when modifying any kernel parameters to reduce the risk of adverse effects.

`sysctl` Used to view or modify kernel parameters at runtime.

Options:	*<string>*	Searches for string provided
	`-a`	Displays all values currently available
	`-w`	Writes (changes) **`sysctl`** value for current session only
	`-p`	Loads **`sysctl`** values from a configuration file (persistent)

Syntax:

```
sysctl <string>
sysctl -a
sysctl –w variable=<value>
sysctl –p <filename>
```

Examples:

```
[root@CentOS6_A ~]# sysctl vm
[root@CentOS6_A ~]# sysctl net.ipv4
[root@CentOS6_A ~]# sysctl -a
[root@CentOS6_A ~]# sysctl –w kernel.domainname=(none)
[root@CentOS6_A ~]# sysctl –p /etc/sysctl.conf
```

`sysctl -a` reveals all possible parameters and their values. Use meta-characters (strings) to refine a search. For instance, **`sysctl net`** lists variables beginning with "*net*" and **`sysctl fs`** lists variables beginning with "*fs*".

Note: The **`sysctl -a`** output prevents the preceding */proc/sys* from being displayed in the results.

`sysctl -w variable=`*<value>* changes or writes variable values for the active session only. The file */proc/sys/net/ipv4/icmp_echo_ignore_all* is equivalent to *net.ipv4.icmp_echo_ignore_all* parameter in */etc/sysctl.conf*. Again, **`sysctl`** assumes the initial */proc/sys/* portion of the path.

To reduce the possibility of a ping flood, also known as an ICMP flood attack, set ICMP ping request value to *ignore* for all. Either of the following two commands have the same results.

```
[root@CentOS6_A ~]# sysctl -w net.ipv4.icmp_echo_ignore_all=1
[root@CentOS6_A ~]# echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_all
```

`sysctl –p /etc/sysctl.conf` updates the values within */etc/sysctl.conf* as if being rebooted. *Sysctl.conf* is an editable file where parameters and values are added, deleted, or modified. By default, the system reads and loads these values every time the system reboots. Changes made to the *sysctl.conf* file are updated at any time using the **`-p`** switch followed by the configuration file. Using the example above, the following entry is made to the */etc/sysctl.conf* file prior to using the **`-p`** option.

```
[root@CentOS6_A ~]# echo “net.ipv4.icmp_echo_ignore_all = 1” >>
/etc/sysctl.conf
[root@CentOS6_A ~]# sysctl -p /etc/sysctl.conf
```

5.3.3 ulimit

In addition to **`sysctl`**, bash provides **`ulimit`** to set limits on system-wide resources for users in that shell. Limits are set to ensure the most important processes continue to run and competing processes do not consume more resources than is good for the system. For example, a user who starts too many processes could make the system unusable for others. A fork bomb, a denial of service (DOS) attack where a process continues to replicate itself until all resources are depleted, may occur if a limit is not placed on running processes. The **`ulimit`** settings can filter max number of processes on user, group, and domain basis. Settings are for the current shell and revert to their default unless permanently changed.

`ulimit`	Sets or prints the limits imposed on files written by the shell and its child processes. It also provides control over the resources available to the shell and to processes started by the shell.		
	Options:	**`-a`**	Displays limits imposed on resources available to the current shell.
		`-c` *<value>*	(core) Displays or sets core file size. If set to zero (0) core dumps are not allowed. Therefore, if process aborts, there is no debugging.
		`-u` *<value>*	(nproc) Displays or sets max number of processes per user.

Syntax: **`ulimit -a`**

`ulimit -c` *<value>*

`ulimit –u` *<value>*

Examples:

```
[root@CentOS6_A ~]# ulimit -a
Core file size          (blocks,  -c)     0
Data seg size           (kbytes,  -d)     unlimited
…
Max user processes               (-u)     6904
Virtual memory           (kbytes, -v)     unlimited
File locks                       (-x)     unlimited
```

```
[root@CentOS6_A ~]# ulimit -c   0
```

```
[root@CentOS6_A ~]# ulimit –u  7000
```

User limits imposed on resources are set in */etc/security/limits.conf*. The following example limits a maximum of three logins for user *bob* on this system.

```
[root@CentOS6_A ~]# more /etc/security/limits.conf
# /etc/security/limits.conf
#
# Each line describes a limit for a user in the form:
# <domain>        <type>            <item>            <value>
# …
# <domain> can be:
#     - a user name
#     - a group name
# …
# <type> Can have two values:
# …
# <item> Can be one of the following:
#     -  core –limits the core file size (KB)
# …
#     - nproc – max number of processes
#     - maxlogins – max number of logins for this user
# …
bob   -     maxlogins   3
# End of file
```

Complete Exercise 8-5 in Student Workbook

Linux Kernel Tuning

5.4 Examining Processes

A process is a program in execution, a running instance of that program. There are two types of processes in Linux: foreground and background. A foreground process is interactive and initialized from the terminal window. A foreground process is started by a user, as opposed to automatically starting at boot time. A background process is non-interactive, does not need user input, and is not connected to a terminal. Processes running in the background are daemons.

Every process runs a single program and has at least one thread. When a process is created, a program counter tracks the next instruction to be executed. The process can create new threads upon execution. A process table maintains the details of these processes.

Processes are created by *fork()* that creates an exact copy of the original process. The copy becomes the child process. The parent and child processes are separate processes. Therefore, if a parent changes any of its variables, the child process does not see these changes and vice versa. By forking a copy of itself, the parent can continue to run its program while having the child take over another part of the job. The child, for instance, could do a print job and when complete, *exit()*.

The system is designed so the parent process waits for the child process to finish and ensures all went well. The child process finishes and its details are kept in the process table until the parent asks for its exit details. The parent does this by calling *wait()*.

Child processes that have not been "*wait()*ed" for are called zombie processes. The zombie does not take up any system resources, but displays when requesting a list of current processes. Figure 9 provides a flowchart of a system call for process control.

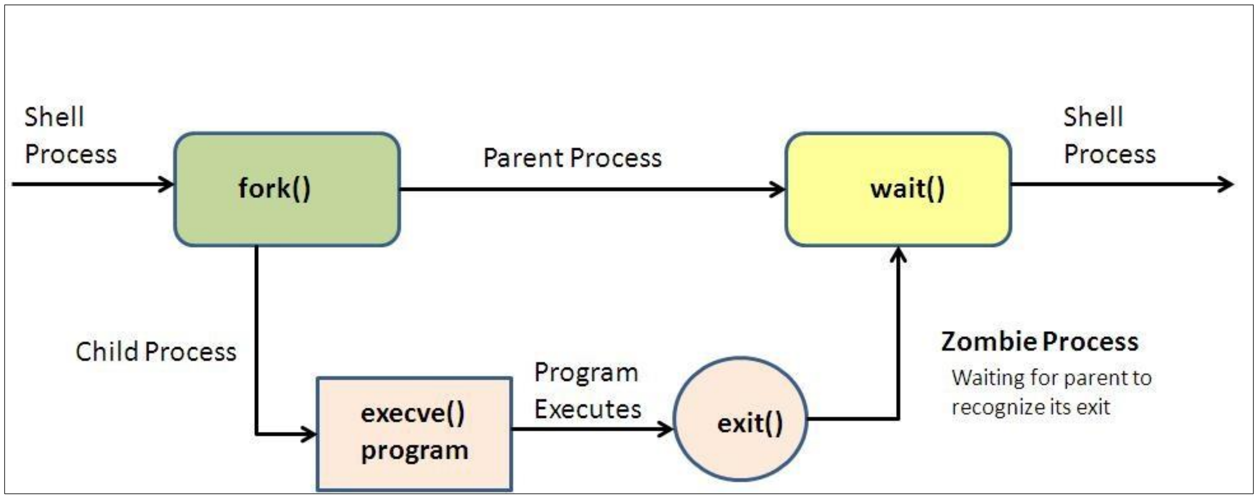


Figure 9. System calls for process control.

5.4.1 Process Properties

Linux provides a variety of tools to determine the state and activity of a process. These tools are invaluable in identifying rogue or abnormal processes. An analyst should monitor Linux processes to ensure only the services and processes desired are running.

The Linux **ps** command displays a snapshot of the current shells' processes. Several options are available to narrow the results and number of fields displayed. Some information available includes: process state, user ID, process ID, parent process ID, and the command that launched the process.

The kernel assigns a unique process identifier (PID) to every process. PIDs are assigned in the order the process is created and begin with '1'. Linux does not supply a system call that creates a new process running a program; instead, a process clones itself to create a new process using the *fork()* system call. As each process terminates, it creates an accounting record describing the resources used. This information is in the process table.

The following is an example of the **ps** command with three often used options (e.g., **-e, -l, -f**):

```
[root@CentOS6_A ~]# ps -elf
F S UID    PID  PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY        TIME CMD
4 S root     1     0  0  80   0 -  4837 poll_s 10:11 ?      00:00:01 [/sbin/init]
1 S root     2     0  0  80   0 -     0 kthrea 10:11 ?      00:00:00 [kthreadd]
1 S root     3     2  0 -40   - -     0 migrat 10:11 ?      00:00:00 [migration/0]
1 S root     4     2  0  80   0 -     0 ksofti 10:11 ?      00:00:00 [ksoftirqd/0]
1 S root     5     2  0 -40   - -     0 cpu_st 10:11 ?      00:00:00 [migration/0]
5 S root     6     2  0 -40   - -     0 watchd 10:11 ?      00:00:00 [watchdog/0]
1 S root     7     2  0  80   0 -     0 worker 10:11 ?      00:00:01 [events/0]
1 S root     8     2  0  80   0 -     0 worker 10:11 ?      00:00:00 [cgroup]
1 S root     9     2  0  80   0 -     0 worker 10:11 ?      00:00:00 [khelper]

<output truncated>

0 S root  2667     1  0  80   0 - 112404 poll_s 10:12 ?     00:00:03 gnome-terminal
4 S root  2668  2667  0  80   0 -  2055 Linux_s 10:12 ?      00:00:00 gnome-pty-helper
4 S root  2669  2667  0  80   0 - 27116 wait   10:12 pts/0  00:00:00 bash
1 S root  3242     1  0  80   0 -  4323 rt_sig 11:01 ?      00:00:00 /usr/sbin/anacron
4 R root  3253  2669  3  80   0 - 27559 -      11:01 pts/0  00:00:00 ps -elf
[root@CentOS6_A ~]#
```

5.4.2 Process States

The **ps** man pages define all header definitions. The current process state is identified under the 'S' heading when running **`ps -elf`**. Listed below are the meanings of the various process states.

State	Meaning
R	Running - Running or runnable (in run queue)
S	Sleeping - Waiting for some resource or event to occur (interruptable sleep)
Z	Zombie - Dead process whose process table entry still exists
T	Stopped - Suspended process (not allowed to execute)

5.4.3 Process Interruption

A process interruption or event may occur in one of two ways:

Interrupt	A signal generated by the hardware when it wants the processor's attention. It usually only needs a short period of CPU time and afterwards the original process can resume execution. An interrupt is caused by some event external of and asynchronous to the currently running process, such as completion of I/O.
Trap	Software written to catch an exception generated by the CPU. A trap is an error or exception condition generated within the currently running process (e.g., illegal access to a file or an arithmetic exception).

Signals are process-level interrupt requests that can be sent in one of three ways:

Terminal	CTRL + C and CTRL + D to kill, interrupt, or suspend processes.
Administrator	With a **`kill`** or **`pkill`** command to get desired results. Syntax: **`kill`** *`[-signal] <PID(s)>`* Example: **`kill -9 19437`** `or` **`pkill -9 cupsd`**
Kernel	When a process commits an infraction such as division by zero.

`kill -l` displays all signals available. Some of the more common signals are:

#	Name	Description
1	SIGHUP	Hangup - restarts a daemon
2	SIGINT	Terminal interrupt - user hits interrupt key. <CTRL-C>
3	SIGQUIT	Quit from terminal - process produces a core dump file
9	SIGKILL	Surest kill - can't be trapped
15	SIGTERM	Default termination signal used by the **kill** command

When a script is running and a user performs an untimely exit by using an interrupt (CTRL+C), the script does not complete and may create a core dump file. Insert a trap into the script to catch the signal and execute something else instead.

Creation of core dump files occurs when an executing process terminates abnormally. Some examples are:

- Receiving a SIGQUIT signal
- Error during process execution
- Program attempt to write beyond the area of memory allocated

The core file contains an image of the process' memory at the time of termination. A debugger can use the image to see the state the program was in when it terminated. Most Linux distributions have core file creation turned off to ensure proper space utilization and security. Use the **`ulimit`** command to configure core file size.

Complete Exercise 8-6 in Student Workbook

Process Management

6 File System Structure

A file system defines the way files are named, stored, organized, and accessed on a logical volume and give the OS a roadmap to access data on a type of file system. File system types are physical, network, and special-purpose. A disk file system randomly addresses data quickly from disk storage media. Multiple processes can promptly access data anywhere on the media without regard to sequential location. Network-based file systems are client/server applications that allow a computer to view and access files on a remote system. The Linux kernel implements special-purpose VFSs that specify an API between a system's kernel and a more concrete file system.

Common Linux disk file systems:

Linux distros commonly use EXT3, EXT4, or XFS disk file systems. EXT(*X)* is a journaling file system, discussed later in this topic.

Common network-based file systems:

Networked File System (NFS)
Server Message Block (SMB)

Common virtual file systems:

swapfs (used to assign addresses for anonymous memory pages "Swapping")
procfs (provides information about processes and other system information)
tmpfs (temporary file storage in memory instead of the hard disk)

The Linux OS resides on multiple special purpose file systems. During boot, these file systems mount as one, and appear as a single file system. The root file system contains the *boot, bin, sbin, etc, dev,* and *tmp* directories that contain files used to keep the system running.

The system may contain additional file systems to store specific data such as system files or user files. Run **fdisk -l** *<device>* to view the disk's physical partitioning scheme.

```
[root@CentOS6_A ~]# fdisk -l /dev/sda

Disk /dev/sda: 21.4 GB, 21474836480 bytes
255 heads, 63 sectors/track, 2610 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot Start       End    Blocks     Id    System
/dev/sda1  *       1        39    307200     83    Linux
/dev/sda2         39      2358    18631680   83    Linux
/dev/sda3       2358      2358    2031616    82    Linux swap / Solaris
[root@CentOS6_A ~]#
```

Note: Root file system is indicated with an (*) denoting it is the bootable partition.

The command **fdisk** *<device>* provides access to the **fdisk** command action feature (help).

```
[root@localhost ~]# fdisk /dev/sda
Device contains neither a valid DOS partition table, nor Sun, SGI or OSF
disklabel.  Changes will remain in memory only, until you decide to write
them.  After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by
w(rite)
Command (m for help):
```

At the `command (m for help):` prompt, type the letter **l** and hit return to see a list of known partition types. Notice the ID and system value that correlates to the partitions revealed in the example below.

```
[root@CentOS6_A ~]# fdisk /dev/sda

Warning: DOS-compatible mode is deprecated.  It's strongly recommended to
         switch off the mode (command 'c') and change display units to
         sectors (command 'u')

 Command (m for help): l

 0  Empty               1e  Hidden W95 Fat1   80  Old Minix          be  Solaris boot
 1  FAT12               24  NEC DOS           81  Minix / old Lin    bf  Solaris
 2  XENIX root          39  Plan 9            82  Linux swap / So    c1  DRDOS/sec (FAT -
 3  XENIX usr           3c  Partition Magic   83  Linux              c4  DRDOS/sec (FAT -
 4  FAT16 <32M          40  Venix 80286       84  OS/2 hidden C:     c6  DRDOS/sec (FAT -
 5  Extended            41  PPC PReP Boot     85  Linux extended     c7  Syrinx
 6  FAT16               42  SFS               86  NTFS volume set    da  Non-FS data
 7  HPFS/NTFS           4d  QNX4.x            87  NTFS volume set    db  CP/M / CTOS / .
 8  AIX                 4e  QNX4.x 2nd part   88  Linux plaintext    de  Dell Utility
 9  AIX bootable        4f  QNX4.x 3rd part   8e  Linux LVM          df  BootIt
 a  OS/2 Boot Manag     50  OnTrack DM        93  Amoeba             e1  DOS access
 b  W95 FAT32           51  OnTrack DM6 Aux   94  Amoeba BBT         e3  DOS R/O
 c  W95 FAT32 (LBA)     52  CP/M              9f  BSD/OS             e4  SpeedStor
 e  W95 FAT16 (LBA)     53  OnTrack DM6 Aux   a0  IBM Thinkpad hi    eb  BeOS fs
 f  W95 Ext'd (LBA)     54  OnTrackDM6        a5  FreeBSD            ee  GPT
 10  OPUS               55  EZ-Drive          a6  OpenBSD            ef  EFI (FAT-12/16/24)

 <output truncated>
 Command (m for help): q

[root@CentOS6_A ~]#
```

The command **df** reveals all Linux file systems, disk space usage, and logical mount points in blocks. Three options are available to view the output of the **df** command. To view the information in human-readable format use **df –h**. To view the output in kilobytes or megabytes use the **–k** or **–m** options.

```
[root@CentOS6_A ~]# df -k
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/sda2             18339256   2692168  14715504  16% /
tmpfs                   506352       264    506088   1% /dev/shm
/dev/sda1               297485     33066    249059  12% /boot
[root@CentOS6_A ~]#
```

6.1 Linux File Systems

Linux uses an ExtX journaling file system which is based on an older Linux File System (UFS). These “journaling” filesystems maintain a journal used to repair inconsistencies that may occur after an improper shutdown. A journaling filesystem writes metadata to the journal that is flushed to the hard disk before each instruction returns. If a crash occurs, the set of updates written to the journal can be fully committed to the file system. This guarantees the structure of the filesystem is always internally consistent. The basic layout information of Ext is stored in the superblock data structure and the group descriptor table, located in the beginning of the file system. They contain the basic size and configuration information of the file system.

6.1.1 Block Group

The file system is organized into block groups containing a superblock and a group descriptor table describing the block group layout. Figure 10 identifies the block group layout:

- Superblock
- Group descriptor table
- Block bitmap (an allocation of blocks used or free)
- Inode bitmap (an allocation of inodes used or free)
- Inode table
- Blocks containing file data

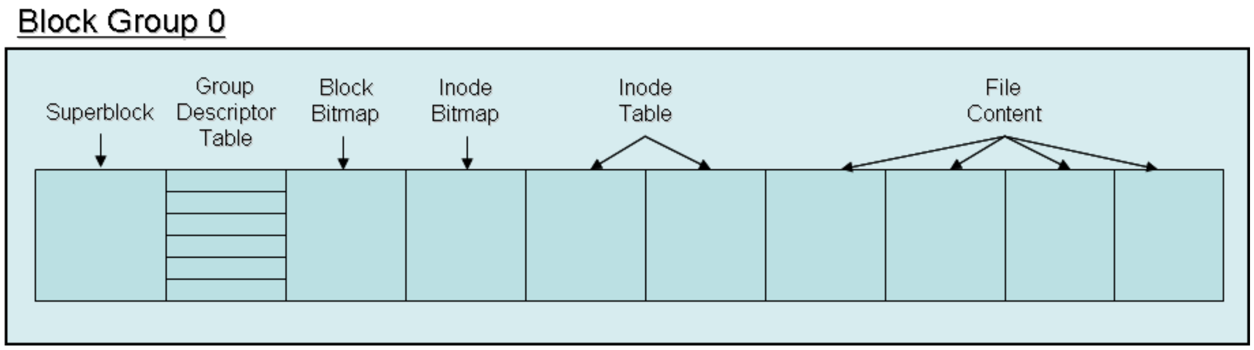


Figure 10. Block Group 0

6.1.2 Superblock

An Ext **superblock** is located 1024 bytes from the start of the file system, is 1024 bytes in size, and contains the following information:

- Total number of inodes
- Total number of blocks
- Block size
- Number of blocks per block group
- Number of inodes per block group
- Number of reserved blocks before the first block group

Many Linux systems do not have boot code in the file system with the kernel. Instead, a bootloader in the disk's MBR is used to locate the kernel during startup.

Backup copies of the primary superblock and group descriptor table exist throughout the file system in case the primary superblock is damaged. The **dumpe2fs** command prints the superblock and block group information for the filesystem. The following command identifies the primary and backup locations of the superblocks of */dev/sda1*.

```
[root@CentOS6_A ~]# dumpe2fs /dev/sda1 | grep -i superblock
dumpe2fs 1.41.12 (17-May-2010)
  Primary superblock at 1, Group descriptors at 2-3
  Backup superblock at 8193, Group descriptors at 8194-8195
  Backup superblock at 24577, Group descriptors at 24578-24579
  Backup superblock at 40961, Group descriptors at 40962-40963
  Backup superblock at 57345, Group descriptors at 57346-57347
  Backup superblock at 73729, Group descriptors at 73730-73731
  Backup superblock at 204801, Group descriptors at 204802-204803
  Backup superblock at 221185, Group descriptors at 221186-221187
[root@CentOS6_A ~]#
```

6.1.3 Group Descriptor Table

The group descriptor table comes immediately after the superblock. This table describes each block group's layout.

- Starting block address of the block bitmap
- Starting block address of the inode bitmap
- Starting block address of inode table
- Number of unallocated blocks in the group
- Number of unallocated inodes in the group
- Number of directories in the group
- Unused bytes

6.1.4 Data Block

A **block** is the basic file allocation unit for EXT(*X*) and is used to store file content. Data block sizes are defined in the superblock. A block is a group of consecutive sectors and can be 1024 bytes, 2048 bytes, or 4096 bytes.

- All blocks are given an address starting with 0. Block 0 is located in the first sector of the file system.
- All blocks belong to a block group. Group 0 begins immediately after file system's reserved blocks.
- Allocation status of blocks is determined using group's block bitmap.

Linux based systems write nulls out to the end of the block, therefore, slack space does not exist.

6.1.5 Inode

The metadata for each file and directory is stored in a data structure called an **inode**. Inodes contain information processes need to access a file. Inodes within each filesystem are the same size. The default for Ext3 is 128 bytes but it may also use 256 bytes. Ext4 uses 256 bytes. The inode value and other filesystem information can be viewed with the **`dumpe2fs`** command mentioned earlier. Each file and directory is allocated one inode. There is an inode table and an inode bitmap in each block group. The allocation status of an inode is determined using the inode bitmap whose location is given in the group descriptor table.

The data structure for an inode contains:

- File size
- File ownership (user, group, other)
- File timestamps (modify, access, change)
- File type (directory (d), file (-), block device (b), character device (c))
- File access permissions (read, write, execute)
- Number of links
- Table of contents containing pointers to data blocks
    - Twelve direct block pointers
    - One single indirect block pointer
    - One double indirect block pointer
    - One triple indirect block pointer

The inode table of contents lists the blocks on the disk where the data is stored. To maintain speed, the size of the table of contents is usually fixed. ExtX is designed for efficiency of small files. Therefore, each inode can store addresses for the first 12 blocks that a file has allocated. These are called direct pointers.

If a file needs more than 12 blocks, a 13th block is allocated to store the remaining addresses. The pointer to this block is called an indirect block pointer.

If a file has more blocks than are addressed by 12 direct block pointers and the indirect block pointer, a double indirect block pointer is used. A double indirect block occurs when the inode points to a block that contains a list of single indirect block pointers, where each points to blocks that contain a list of direct pointers. If a file needs more space, a triple indirect block pointer is used. A triple indirect block contains addresses of double indirect blocks that contain addresses of single indirect blocks. Figure 11 outlines the inode structure.

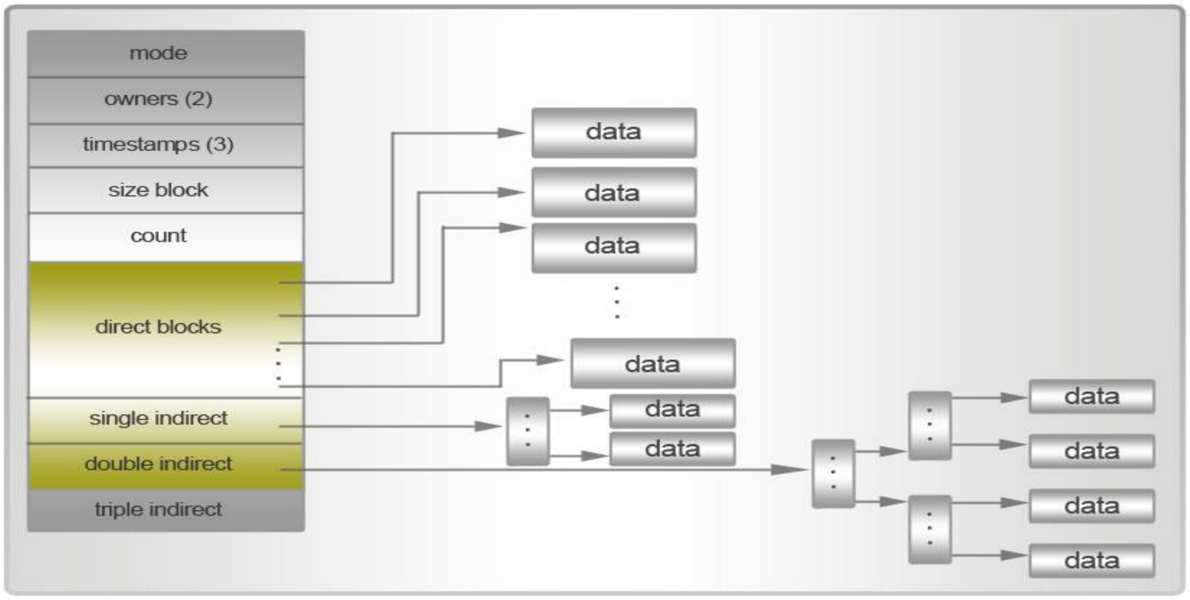


Figure 11. Inode structure overview.

Ext3 systems support three timestamps. Ext4 also supports a creation time which is not discussed here. Only the last value of each time is stored.

Modification	(M-time) Time file's data layer was last modified.
Access	(A-time) Time file's data layer was last accessed.
Change	(C-time) Time file's inode (metadata layer) was last changed/created.

The **stat** command is used to view file or file system status to include timestamps.

```
[root@CentOS6_A ~]# stat /tmp/file1.txt
File: `/tmp/file1'
Size:  27           Blocks:  8  IO Block:  4096   regular file
Device: fd00h/64768d Inode:  83232     Links:  1
Access: (0644/- rw-r--r--) UID:  ( 0/ root)  GID:  ( 0/ root)
Access: 2014-09-15 10:23:31.768684839  -0500
Modify: 2014-09-15 10:22:38.248778278  -0500
Change: 2014-09-15 10:22:38.248778278  -0500
```

The **touch** command creates an empty file and can also alter time stamps. Modification and access times are changed individually with the **-m** and **–a** options. The **–t** option changes modification and access times simultaneously.

```
[root@CentOS6_A ~]# touch a.txt
[root@CentOS6_A ~]# stat a.txt
File: `a.txt'
Size:  0       Blocks:  0  IO Block:  4096   regular empty file
Device: 802h/2050d   Inode:  130404    Links:  1
Access: (0644/- rw-r--r--) UID:  ( 0/ root)  GID:  ( 0/ root)
Access: 2017-06-27 09:02:02.880349957  -0700
Modify: 2017-06-27 09:02:02.880349957  -0700
Change: 2017-06-27 09:02:02.880349957  -0700
[root@CentOS6_A ~]# touch –t 201706270904 a.txt
[root@CentOS6_A ~]# stat a.txt
File: `a.txt'
Size:  0       Blocks:  0  IO Block:  4096   regular empty file
Device: 802h/2050d   Inode:  130404    Links:  1
Access: (0644/- rw-r--r--) UID:  ( 0/ root)  GID:  ( 0/ root)
Access: 2017-06-27 09:04:00.000000000  -0700
Modify: 2017-06-27 09:04:00.000000000  -0700
Change: 2017-06-27 09:05:19.768609717  -0700
```

6.1.6 Directory Entry

Directory entries give the hierarchical structure of the file system. A **directory entry** is a simple data structure containing filename and inode address where the file's metadata can be found. It also contains a pointer to the next entry. Figure 12 breaks down a directory entry structure.

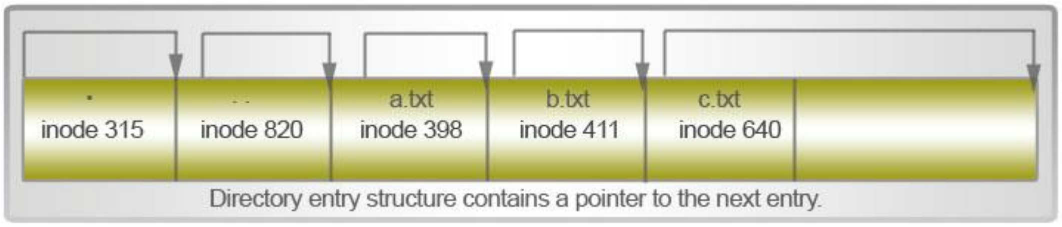

.
inode 315
..
inode 820
a.txt
inode 398
b.txt
inode 411
c.txt
inode 640
Directory entry structure contains a pointer to the next entry.

Figure 12. Directory entry.

When a file or directory is deleted, the filename needs to be modified so it is not printed by the OS. The OS hides the file/directory by increasing the record length of the previous directory entry so it points to the entry after the one being hidden. See Figure 13 below.

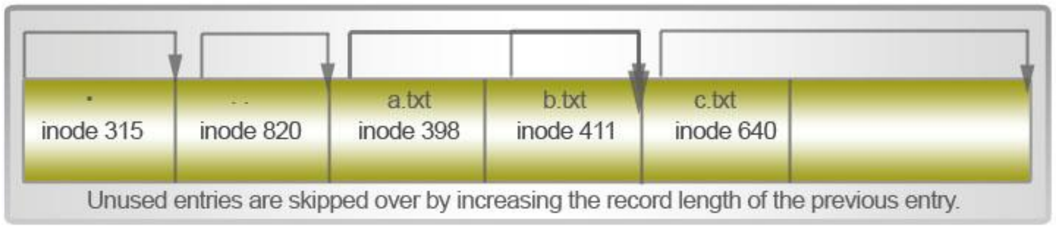

.
inode 315
..
inode 820
a.txt
inode 398
b.txt
inode 411
c.txt
inode 640
Unused entries are skipped over by increasing the record length of the previous entry.

Figure 13. Linux deleted files.

6.1.7 File and Directory Linking

Since inodes do not contain filename information, a file can have more than one filename, or link, to the same inode. A file's normal link count is one and is displayed in a long directory listing. When a hard link is created, the inode updates its link count to reflect and the file is then accessed through any of its links. Use `ls –il` to see if a file has any additional links.

A **hard link** is a pointer to a file's inode and is the same as the original file. The syntax is as follows:

`ln` *<target file/dir> <link name>*

To create a hard link to the file *testfile2* from *testfile1* use the following command:

```
[root@CentOS6_A ~]# ln testfile1 testfile2
```

Perform `ls -il` on the two files. Notice the inode number is the same for both files. The file named *testfile2* is considered an alias name for the file *testfile1.* There is now one file with two filenames.

A **symbolic link** is a file containing the pathname of another file. With a symbolic link, a file's contents are accessed through that pathname. A symbolic link is a pointer to the file's name. Unlike a hard link, the link count does NOT increase when a symbolic link is created. To create a symbolic link, use the following command:

```
[root@CentOS6_A ~]# ln -s test1 test2
```

Perform `ls -il` on the two files. Notice the *test1* inode number in this file is different from the *test2* inode number. In addition, notice size and permissions of the two files are different.

Below is an example of the symbolic link creation and comparison.

```
[root@CentOS6_A tmp]# ln -s /tmp/test1 /home/test2
[root@CentOS6_A tmp]# ls -lisa /tmp/test1 ; ls -lisa /home/test2
796572 4 -rw-r--r--. 1 root root 13 Mar 23 06:36 /tmp/test1
130716 0 lrwxrwxrwx. 1 root root 10 Mar 23 06:46 /home/test2 -> /tmp/test1
[root@CentOS6_A tmp]#
```

When a directory is moved to a new location, users must navigate to the new location and find their files. Symbolic links take care of this problem by pointing to the new location. In the example above, the file *test2* is symbolically linked to *test1* and the file type is now indicated with the letter '**l**', in the first position of the permission field.

7 Operations on File Systems/Directory Structure

The OS module introduced the file system hierarchy of Windows and Linux OSs. Both OSs have a hierarchy standard that defines the preferred directory structure and directory contents. For Linux distributions, the Linux Foundation maintains the preferred structure and expects all UNIX-based systems to adhere to this format. Accessing directories and files on a file system requires first being able to identify and access the storage devices or disk device. Disk devices and naming conventions associated with Linux are introduced in this section.

7.1 Accessing Disk Devices

Disk devices are named based on use: physical or logical. A physical device consists of its bus and device address. A logical device is used with file system commands when referring to a device. During boot, a device hierarchy is created from devices connected to the system. The kernel uses this hierarchy to map associated drivers to their devices. A device file system (devfs) manages devices on the system and creates logical links in */dev*.

Logical device names are used to access disk devices when performing any of the following:

- Adding a new disk to the system
- Moving a disk from system to system
- Mounting or accessing a file system on a local disk
- Backups
- Repairing a file system

The **mount** command with no options displays a list of mounted filesystems as shown below.

```
[root@CentOS6_A ~]# mount
/dev/sda2 on / type ext4 (rw)
proc on /proc type proc (rw)
/dev/sda1 on /boot type ext4 (rw)
/dev/sr0 on /mnt/cdrom
[root@CentOS6_A tmp]# ls -lisa /tmp/test1 ; ls -lisa /home/test2
796572 4 -rw-r--r--. 1 root root 13 Mar 23 06:36 /tmp/test1
130716 0 lrwxrwxrwx. 1 root root 10 Mar 23 06:46 /home/test2 -> /tmp/test1
[root@CentOS6_A tmp]#
```

mount also maps logical devices to mountpoints on a file system. Below, the logical device name, *</dev/sda3>*, maps to directory or mountpoint, *</media>*, on the local system.

```
[root@CentOS6_A ~]# mount /dev/sda3 /media
```

Table 3 describes naming conventions for logical disks and devices in Linux**.**

Table 3. Disk Naming Conventions.

Filename	Description
/dev/sda	First SCSI drive
/dev/sda1	First SCSI disk(a), first partition (1)
/dev/sdb3	Second SCSI disk(b), third partition (3)
/dev/hda	Master disk(a) on Primary IDE Controller
/dev/hdb	Replica disk(b) on Primary IDE Controller
/dev/sr0	CD-ROM

7.2 File Types

Linux classifies virtually everything as a file. Directories, devices, processes, and some APIs are considered files. There are two key file types, regular and special.

Regular/Ordinary	Contains only data as a stream of characters.
Text	File containing only printable lines of characters.
Binary	Contains printable and non-printable characters. (Entire ASCII range). Most commands, ELF (Executable and Linkable Format).

Other regular files include image (jpg), compressed (bzip), and audio (au) files.

Special	Special files are divided into multiple sub-categories based on their function.
Directory	Contains other directories and files. Directories do not contain data as a stream of characters, but maintain information about its subdirectories and files. Identified by a 'd' in the mode field of `ls -l`.
Block device	Data is read and written in blocks using a buffer cache. Examples are memory or a disk (e.g., /dev/ram0, /dev/sda). Identified by a 'b' in the mode field of `ls -l`.
Character device	A raw device. Reads and writes data as a stream of characters. Examples are a printer or console (e.g., /dev/lp0 or /dev/console). Identified by a 'c' in the mode field of `ls -l`**.**
Named Pipe	Connects output of one process to the input of another. Identified by a 'p' in the mode field of `ls -l`.
Symbolic Link	Stored as a textual representation of the referenced file's path. Identified by an 'l' in the mode field of `ls -l`.
Socket	Used by IPC to communicate between 2 processes. Identified by an 's' in the mode field of `ls -l`**.**

The **file** command identifies file type by probing each object with three types of tests until one succeeds. The file type identifier is located in the header line of a file.

```
[root@CentOS6_A ~]# file /etc/passwd
/etc/passwd: ASCII text
[root@CentOS6_A ~]# file /usr/bin/passwd
/usr/bin/passwd: setuid ELF 64-bit LSB shared object, x86-64, …
```

Complete Exercise 8-7 in Student Workbook

File and Directory Linking and File Type Recognition

Objectives

Day 4

Upon completion of this training day, students will:

- ❖ Identify shell and environmental variables and make their values persistent.
- ❖ Create a simple BASH script.
- ❖ Interpret outcome of PowerShell Core combined with Linux commands and functions.
- ❖ Interpret login, non-login, restricted, and secure shells, and their functions.
- ❖ Use ssh to securely access, copy, and transfer files, remotely.
- ❖ Explain passwd, shadow, and group files and their vulnerabilities.
- ❖ Demonstrate the skills necessary to manage user and group accounts.

Exercises

This training day includes the following exercises:

- ❖ Exercise 8-8, Create a Bash Script and Use Functions in a PowerShell Script
- ❖ Exercise 8-9, Manage Login Shells and Scripts
- ❖ Exercise 8-10, Creating and Managing User and Group Accounts

JCAC-eTC

Complete self-study activities and assignments in the online training environment.

8 Variables

Variables store information used later by a script or program. They provide a way of labeling data using a descriptive name a program can clearly understand. Variable names are preceded by the operator '$', that translates the variable name into its contents. A variable and its value are displayed using the **echo** command (i.e., **echo $PATH**). Two types of variables are shell (local) and environment (global). Environmental variables are further broken down as either user-defined or system variables. System variables are generally indicated as upper-case, i.e., PATH, HOME, and USER.

8.1 Shell Variables

Shell variables are local because they are only accessed within the shell where they were created. A locally created variable does not, by default, transfer to a newly opened shell because it is used by the shell itself, not its child processes or applications. A new shell is invoked by opening a new terminal window, starting a shell (*/bin/bash*), or switching users from the CLI.

Operating Systems	A shell is a program that provides text only interface for Linux and other UNIX like OSs. The shell presents each user with a prompt, executes user commands, and supports a custom environment for each user.

The **set** built-in shell command displays existing shell variables, environmental variables, and shell functions. These values are not passed to child processes and applications by default.

```
[root@CentOS6_A ~]# set
BASH=/bin/bash        (This shell type)
COLUMNS=61            (Width of current shell window, changes if window is
resized)
LINES=25              (Height of current shell window, changes if window is
resized)
PS1='[\u@\h \w]\$ '   (Shell prompt - [root@CentOS6_A ~]#
Student=bob           (User defined variable, lower-case if not an
environmental variable)
<output truncated>
[root@CentOS6_A ~]#
```

Creating a variable and assigning it a value requires the '=' operand. The following example illustrates creating a local variable, proving it is only available locally, and displaying the variable.

```
[root@CentOS6_A ~]# name=mike
[root@CentOS6_A ~]# echo $name
mike
[root@CentOS6_A ~]# set | grep name
name=mike
[root@CentOS6_A ~]# /bin/bash                (This is a new shell)
[root@CentOS6_A ~]# echo $name               (No results, new shell)
[root@CentOS6_A ~]# set | grep name          (No results, new shell)
[root@CentOS6_A ~]# exit                     (exit second shell)
```

A variable can be exported to make it available every time a new shell is invoked from the parent. The variable transfers to a new shell for that session only. After a variable is created, run **`export`**, as shown below, followed by the variable name to allow child processes to access the variable. **`Export`** is a bash built-in command that marks an environment variable to be exported with any newly forked child processes. It allows a child process to inherit all defined variables.

```
[root@CentOS6_A ~]# course=jcac
[root@CentOS6_A ~]# export course
[root@CentOS6_A ~]# echo $course
jcac
[root@CentOS6_A ~]# /bin/bash              (This is a new shell)
[root@CentOS6_A ~]# echo $course
jcac                                       (Success, exported for child
                                           processes)
[root@CentOS6_A ~]# set | grep course
course=jcac                                (Success, exported this
                                           session only)
[root@CentOS6_A ~]# exit                   (exit second shell)
```

The *$PATH* variable searches for files in response to user commands. *$PATH* is viewed below.

```
[root@CentOS6_A ~]# echo $PATH
/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/bin:/bin:/root/bin
```

To totally wipe out and replace the current *$PATH*:

```
[root@CentOS6_A ~]# PATH=/usr/xpg4/bin
[root@CentOS6_A ~]# echo $PATH
/usr/xpg4/bin
```

To append */IT* to the existing *$PATH*:

```
[root@CentOS6_A ~]# PATH=$PATH:/IT
[root@CentOS6_A ~]# echo $PATH
/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/bin:/bin:/root/bin:/IT
```

Once the shell is closed, modifications are lost. Making local variables available after a restart requires updates to login shell scripts, discussed later in this topic.

8.2 Environment Variables

Environment variables are available in most shells. Bash also maintains built-in reserved variables with default settings. These are global, affecting the users' total environment, and are often upper-case for quick identification.

The **env** command displays only existing environmental variables. These values are passed to the shells child processes. Below is a partial variables list using the **env** command.

```
[root@CentOS6_A ~]# env
ORBIT_SOCKETDIR=/tmp-root
HOSTNAME=CentOS6_A
TERM=xterm
SHELL=/bin/bash
MAIL=/var/spool/mail/root
PATH=/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/bin:/bin:/root/bi
n
<output truncated>
[root@CentOS6_A ~]#
```

Creating or modifying variable values from the shell does not automatically make the variable a part of the global environment. Exporting the variables to a startup script or profile is a way of making the local variable a global variable.

User startup scripts and profiles are normally located in the users' home directory and the script name may vary based on type of shell used. Modify *~/.bash_profile* or *~/.bashrc* scripts to make the variable part of the user's global environment. These scripts must be executed again after modification, normally via reboot. An alternative to rebooting is bash's **source** command. **Source** reads and executes the contents of a file that is passed as an argument in the current shell script.

Below, the */IT* directory is appended to the PATH statement by editing the user's *.bash_profile* file. The **export** command makes the variable a part of the user's global environment after the script is executed or sourced.

```
PATH=$PATH:$HOME/bin:/IT
export PATH
```

The following example ensures the new path statement is read and exported by using the **source** command.

```
[root@CentOS6_A ~]# source .bash_profile
[root@CentOS6_A ~]# echo $PATH
/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/bin:/bin:/root/bin:/IT
[root@CentOS6_A ~]#
```

Another example of an environment variable is *$HOME*, the path to a user's home directory.

```
[root@CentOS6_A ~]# echo $HOME
/root
```

The **cd** command with no arguments or options reads *$HOME* and places the ~ symbol in the prompt. The ~ is an alias for the current user's home directory.

```
[root@CentOS6_A /etc]# cd
[root@CentOS6_A ~]#
```

9 Creating Simple Shell Scripts

A shell script is a list of commands, often sequentially executed, containing logic allowing code to be executed under specific conditions. The scripts introduced in the RC scripts section are a perfect example of scripts being used to automate stopping or starting of services at various run-levels.

For a user to create a simple shell script, commands typed at the CLI independently, such as **date** and **whoami**, are combined into one file and executed as a script. Scripts should initially contain a shebang line, identifying the script as a bash script. The shebang line is entered as **#!/bin/bash**. Other shells may be specified, such as **#!/usr/bin/perl** if the script contains Perl code. If shebang is omitted, the script code is executed by the shell used in the parent shell. As discussed previously, when a parent shell environment starts a script, execution of the commands that follow occurs in a subshell.

The '**#**' in shebang does not represent a comment, but rather the start of the script. All other lines that begin with a '#' are comment lines. Comments should be used at the beginning of the script to explain script details and at the beginning of each subsection to describe what is happening.

User-defined and system variables are used within a script to store and retrieve data from memory. A user may set the variable directly, such as *name=mike* or use an existing system variable such as *$HOME*, which is the home directory path. A "command variable" such as *day=$(date)* may also be created. This creates a user variable, day in this example with a system command. Notice when the **date** command is used as a variable, the format is different. The variable is read by placing its name, preceded by **$,** anywhere in the script. The script checks for existing variable names prior to running each line in the script. Every variable name identified is replaced with its value and the line of code is then executed. The script continues to the next line until finished.

Use the vi editor to create a new script named *welcome.sh* and insert the following content:

```
#!/bin/bash
greeting="Welcome"
user=$(whoami)
day=$(date +%A)(%A displays the weekday name e.g. Sunday)
echo "$greeting back $user! Today is $day, which is the best day of the
entire week!"
echo "Your Bash shell version is: $BASH_VERSION"
```

Note in the second line, the variable *greeting* was declared and assigned a string value of *Welcome*. On the third line, the current user was called by using the **`whoami`** command. This is done through command substitution, meaning the output of **`whoami`** is directly assigned to the variable *user*. The same applies on line four, for the *day* variable. The **`echo`** command sends standard output to the screen. This script can be referenced in *~/.bashrc* and is applied to each new terminal window. The *.bashrc* script is discussed in more detail in the following topics.

After creating the script, make it executable by running the command **`chmod +x welcome.sh`**. The *welcome.sh* script is executed by either running **`bash welcome.sh`** or, if the file is stored in a location not in the path statement, running **`./welcome.sh`** from the directory it resides in or the absolute path.

10 Functions within a PS Script

A PS script is essentially a text file containing a series of PS commands. Scripts are identified by a *.ps1* extension and are executed through an interactive console using the script name. Powershell Core, *pwsh.exe,* is cross-platform, meaning it runs on several *Nix variants, and runs side-by-side with Windows PS (*powershell.exe*). PS is currently installed on the *CentOS7_A* VM.

After executing **pwsh**, text and background colors may be reversed, making commands difficult to view. For clarity, those settings can be changed. From the terminal menu, choose *Edit*, then *Preferences* from the terminal window banner. Choose *Colors* from the Preferences window and under the *Text and Background Color* title, clear the check box from *Use Colors from system theme*, and close the *Preferences* window.

Cmdlets may be combined with other cmdlets, functions, arguments and even Linux commands to automate tasks. See the example of a partial PS script below and identify what each line accomplishes.

```
if(!(Test-Path ‘/tmp/pslist’))
{
      New-Item -Path ‘/tmp/pslist’ -ItemType Directory
}
Set-Location ‘/tmp/pslist’
$procs=Get-Process
$procs | Out-File -FilePath ‘/tmp/pslist/procs.txt’
```

The partial script above checks for the existence of a directory and creates the directory if not found. It then creates a file with a list of running processes and places the results into the directory.

If the script was not executing in a PS environment, the first line would be **#!/usr/bin/pwsh**. Cmdlets used in the sample script are listed below:

• **Test-Path**	Determines if all elements of a path exist and if the path sytax is valid.
• **New-Item**	Creates a new item and sets its value. For example, files and directories.
• **Set-Location**	Sets the working location to specified location. A path location.
• **Get-Process**	Gets the processes running on the computer.
• **Out-File**	Writes created output to a file.

A final cmdlet to view the contents of procs.txt would be:
Get-Content /tmp/pslist/procs.txt

10.1 Creating a Simple PowerShell Script

The PS script lines introduced below may be created in **vi** and then executed after the execute permission is set. If creating the script, the file name should end in *.ps1*. Analyze the contents and flow of the script to identify the output.

To accept user input from within a script, use the **`Read-Host`** cmdlet in conjunction with the **`Prompt`** parameter. **`Read-Host`** pauses execution and receives input. **`Prompt`** gives the script user instructions on requested input. The following prompts a user for their age and stores the result as the *$A* variable.

```
$A = Read-Host -Prompt “How old are you?”
```

After receiving the input from the user, the string stored in the variable can be returned to the display using the **`write-host`** cmdlet.

```
Write-Host “Your age is $A”
```

There are a variety of ways to obtain all or part of a cmdlet result and store the data as a variable in a script. The **`Get-Date`** cmdlet is similar to the Bash **`date`** command. **`Get-Date`** uses the *-Uformat* parameter to extract results from specific fields. The following creates a variable *$B* to hold the data extracted from the *%a* field - the day of the week in abbreviated form (i.e., Wed). Other fields are *‘%A’*, full day of the week, and *‘%T’*, time of day in hh:mm:ss format.

```
$B = Get-Date -Uformat ‘%a’
```

PS scripts use the **`Start-Sleep`** cmdlet followed by the *-seconds* option and the number of seconds when a pause in the script is required. This may be needed for error detection or to provide a gap between execution of cmdlets. For example, to have the script pause for 5 seconds before the next line of code is executed add the following line:

```
Start-Sleep -seconds 5
```

Complete Exercise 8-8 in Student Workbook

Create a Bash Script and Use Functions in a PowerShell Script

11 Login and Non-Login Shells

Many shells are available to each OS and they may be either **login** or **non-login** shells. The last field in each line of */etc/passwd,* normally specifies the user's assigned login shell. If not assigned, Linux assigns a default *bash* shell. *Bash* may be invoked as a login or non-login shell.

A login shell requires initial system authentication. During an initial login, the system launches the user's ~/*.bash_profile* shell script. Starting a subsequent shell (sub-shell) from the current shell launches the user's *~/.bashrc* script. This non-login shell requires no additional system authentication. The **echo** command followed by the *$SHELL* argument identifies the current shell.

```
 [root@CentOS6_A ~]# echo $SHELL
/usr/bin/bash
```

If the result is the shell name not prepended by a dash as shown above, it is a non-login shell. If the result is the shell name prepended by a dash as shown below, it is a login shell.

Using **ssh** to login to the system is one example of creating a login shell.

```
 [root@CentOS6_A ~]# echo $0
bash
```

If the output is the name of the shell not prepended by a dash as displayed above, it is a non-login shell. If the output is the name of the shell prepended with a dash, it is a login shell, as shown below. Using **ssh** to login to the system creates a login shell.

```
[root@CentOS6_A ~]# ssh student@192.168.0.80
student@192.168.0.80's password:
Last login: Tue Dec 12 10:37:06 2016 from localhost
[student@CentOS6_A ~]# echo $0
-bash
```

11.1 Login shell

System login begins with the execution of ***/bin/login***, which reads the */etc/passwd, /etc/shadow*, and */etc/group* files for authentication and authorization. A login shell, also referred to as an interactive login is the first process executed under the user ID when logging into a session.

An interactive login shell is also used with remote logging such as **ssh** or **su –**, where new authentication is required. They are interactive because they require authentication with a login and password. The **su** – logs the user in completely with that user's full environment. An **su** alone only switches to the new user, providing a normal shell with an environment inherited from the previous user.

Several startup programs and scripts set up the user's profile and system wide environment. The next few sections discuss the login scripts defined in the following steps.

When invoking *bash* as a login shell, the following steps occur:

Step 1:	A login calls */etc/profile*	System wide environment and startup programs. Default settings for ALL users when starting a login shell.
Step 2:	*/etc/profile* calls scripts in */etc/profile.d*	Scripts contain app-specific startup files.
Step 3:	*/etc/bashrc* script runs	Defines defaults for ALL users when starting a sub-shell. Applies to login and non-login shells.
Step 4:	*~/.bash_profile* calls	User specific environment and startup programs. Specific settings for one user applied when starting a login shell.
Step 5:	*~/.bashrc*	User specific aliases and functions. Specific settings for one user applied when starting a sub-shell. Applies to login and non-login shells.

Examples of *bash* login shells include:

- Shells created using **su** with the '-'

  [root@CentOS6_A ~]# **su -** *<USERNAME>*
- A virtual terminal after a successful login

CTRL+ALT+F2	Opens a virtual terminal in VMware
ALT+F1	Closes the virtual terminal
- Secure shell using **ssh**

  [root@CentOS6_A ~]# **ssh** *<USERNAME>***@***<HOSTNAME or IP>*

11.1.1 SSH Suite

The SSH suite of software contains programs that create a secure, encrypted channel of communication for data exchange between networked devices. In addition to using **ssh** to securely access a remote shell, the suite also offers other secure programs. Two of these are Secure Copy (**scp**) and Secure Shell File Transfer Protocol (**sftp**).

Scp securely copies files over the network. **sftp** is an ftp-like client used for file transfer over a network. Neither program requires a dedicated daemon since they both connect to sshd servers on port 22 by default. Examples of **ssh, sftp**, and **scp** are explained below.

SSH

Using **ssh** requires a successful login to the remote system, after which a shell opens to execute commands on that system. The prompt displays the shell for user and host on the remote machine.

```
[root@CentOS6_A ~]# ssh student@CentOS6_B
student@localhost's password:
[student@CentOS6_B ~]$
```

A shell command can also be executed on the remote machine without obtaining a shell terminal window on that machine. Note the prompt does not change, the command is simply running on the remote machine.

```
[root@CentOS6_A ~]# ssh student@CentOS6_B ls -l /home
student@CentOS6_B's password:
total 8
drwx------. 26 student student 4096 Mar 31 13:06 student
[root@CentOS6_A ~]#
```

SCP

Use **scp** to securely copy files across a network. Most OSs adhere to the syntax and options provided with ssh capabilities. Listed below are a few options and examples.

scp Copies files between network hosts using the same authentication and providing the same security as **SSH**. A password may be required.

Options:	**-r**	Recursively copy entire directories
	-P	Specifies a different port to use, instead of the default port 22. Referred to as port forwarding, this requires modification to the */etc/ssh/sshd_config* file. Restarting the sshd service is required.

Examples:

Copy *test.txt* from a remote host to the local host:

```
[student@CentOS6_A ~]$ scp student@CentOS6_B:test.txt  /home/student
```

Copy *test.txt* from local host to a remote host:

```
[student@CentOS6_A ~]$ scp test.txt  student@CentOS6_B:/home/student
```

Copy the *Study* directory from the local host to the remote host *Notes* directory:

```
[student@CentOS6_A ~]$ scp  -r Study  student@CentOS6_B:/home/student/Notes
```

Copy *mynotes* from the local host to a remote host using port 2222:

```
[student@CentOS6_A ~]$ scp  -P 2222 mynotes  student@CentOS6_B:/home/student
```

SFTP

sftp is similar to **scp** in that it is used for copying files. However, during **sftp** transfer, each packet must be acknowledged (ACK). This acknowledgement, along with the fact that it does not use the FTP daemon for connections, provides for a more secure process, even if it is slower. In addition to copying files, **sftp** also provides options to list, create, and delete directories and files. Remote print working directories can also be listed. Once an **sftp** session is successfully started, a shell with an **sftp>** prompt is displayed as shown:

```
[root@CentOS6_A ~]#  sftp student@CentOS6_B
Connecting to CentOS6_B
student@CentOS6_B's password:
sftp>
```

Type “help” or “?” to view available interactive commands. A truncated list is displayed below.

ls [path]	Display remote directory listing
lls [path]	Display local directory listing
cd path	Change remote directory to ‘path’
lcd path	Change local directory to ‘path’
get remote-path [local-path]	Download file
put local-path [remote-path]	Upload file
quit	Quit sftp (exit also works)
pwd	Print remote working directory
lpwd	Print local working directory

Listed below are examples of some `sftp` commands after initializing the session.

```
sftp> lpwd
Local working directory: /root
sftp> pwd
Remote working directory: /home/student
sftp> put .cshrc
Uploading .cshrc to /home/student/.cshrc
.cshrc
sftp> lcd /tmp
sftp> get .bash_history
Fetching /home/student/.bash_history to .bash_history
.bash_history
sftp>quit
[root@CentOS6_A ~]#
```

11.1.2 Restricted Secure Shell

A restricted secure shell, `rssh`, is a part of the SSH suite. It can be configured to specifically restrict use to `scp` or `sftp`. It forces users to securely copy or transfer files without providing them a shell, as opposed to making them optional. The **scp** performs a faster transfer but only transfers files. It does not list or remove remote directories, nor does it provide any of the other features of **sftp**. However, **scp** doesn’t authenticate every packet, but rides on top of **ssh**. The **sftp** is developed from **scp**.

If not assigned a restricted shell, users with proper permissions may change the current shell by entering the shell executable. Linux provides the **chsh** command. Using **chsh** automatically updates the */etc/passwd* file with the changed shell. For example, the following command changes *billy’s* login shell to the *Bourne* shell.

```
[root@CentOS6_A ~]# chsh -s /bin/sh billy
Changing shell for billy.
Shell changed.
[root@CentOS6_A ~]# su billy
sh-4.1 $ echo $SHELL
/bin/sh
sh-4.1 $
```

11.2 Non-login shell

A non-login shell starts without requiring a new login. When opening a terminal window in an existing GUI session (X terminal), the user automatically receives a session or window manager instead of a login shell.

A Bourne shell reads the user's *~/.profile* script to set the user's environmental variables that subprocesses then inherit.

Users of the Bourne Again (*bash*) shell could use *~/.profile* in a compatible way by executing it explicitly from *bash* specific scripts. But, to keep *bash* backwards compatible with *sh*, the system omits *~/.profile* and instead reads the user's *~/.bash_profile*.

When invoking *bash* as a non-login shell, the following scripts set the shell environment.

Step 1:	A non-login shell calls *~/.bashrc*	User specific aliases and functions. Specific settings for one user applied when starting a sub-shell.
Step 2:	*~/.bashrc* calls */etc/bashrc*	System wide functions and aliases
Step 3:	*/etc/bashrc* calls scripts in */etc/profile.d*	Scripts containing app-specific startup files

Examples of *bash* non-login shells include:

- Shells created using **su** without the "-"

  [root@CentOS6_A ~]# **su** *<USERNAME>*

- Graphical terminals (Simply opening a terminal window without re-logging in)
- Executed scripts

11.3 Login Scripts and Profiles

Regardless of the type of shell used, during enumeration three files are commonly checked due to the individualized information they contain. Two of the files have to do with the users profile, *~/.bash_profile* and *~/.bashrc*. The third file, *~/.bash_history*, contains a list of commands run by the user.

A user can customize their environment after logon by changing the contents of their personal profile, *~/.bash_profile*. *~/.bash_profile* contains user specific environment and startup programs.

```
[root@CentOS6_A joe]# more .bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin

export PATH
[root@CentOS6_A joe]#
```

Some user specific environment and startup programs for ~*/.bash_profile* are:

1 User's shell search path.
2 Path to the user's mail file.
3 User's usenet news server.
4 User's search path for man pages.
5 User's default printer.
6 Sets user's default file creation permissions.
7 Sets listed environment variables.

The example below details default settings that may be found in ~*/.bash_profile*.

1 PATH=$PATH:$HOME/bin:/usr/local/bin:/usr/ccs/bin:
2 MAIL=/var/mail/$LOGNAME
3 NNTPSERVER=server1
4 MANPATH=/usr/share/man: /usr/local/man
5 PRINTER=printer1
6 umask 022
7 export PATH MAIL NNTPSERVER MANPATH PRINTER

~/.bash_profile works in conjunction with a script named *~/.bashrc* that contains user specific aliases and functions. When a user logs onto the system, instructions from these two files execute and create the system and user environments.

```
[root@CentOS6_A joe]# more .bashrc
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . etc/bashrc
fi

# User specific aliases and functions
[root@CentOS6_A joe]#
```

Note: *~/.bash_profile* is similar to a Windows environment, where a new user gets a *Default User* profile when they first logon and afterwards receives their own user profile (*ntuser.dat*).

By default, the *~/.bash_history* file name is stored in the *HISTFILE* variable and saves the most recently executed commands when *bash* exits. These commands are viewed using **echo $HISTFILE**. **history** is a great tool for debugging or for viewing previously executed commands. After viewing a historical list of commands, a `!' followed by the command's corresponding number executes the command at that number. This is useful when there is a need to execute a previous, possibly syntactically long, command.

```
[root@CentOS6_A ~]# history
   332  ls -l
   333  pwd
   334  find / -path /proc -prune -o -perm -2 -exec ls -l {} \;
[root@CentOS6_A ~]# !334
```

11.4 Restricting Shell Access

To restrict user activities and help protect the system from unauthorized access to a shell, login or non-login, the administrator may assign a restricted shell. Linux has the ability to assign a restricted *bash* shell, */bin/rbash*, if available on the system. Restricted shell configurations limit a user's ability to:

- Use **cd** command
- Set or change value of *PATH*, *ENV*, or *SHELL* variables
- Specify a command or filename containing a slash (/) (current directory only)
- Use output redirection (> or >>)

Start the *rbash* restricted shell as follows. Attempt the **cd** command or redirection to a file.

```
[root@CentOS6_A ~]# bash -r
```

11.5 Disabling Shell Access

Linux uses a nologin shell (*/sbin/nologin)* to prevent unauthorized users from obtaining shell access to either a login or non-login shell. Disabled accounts and accounts needing other user level access such as FTP, POP3, and SMTP do not always need a login shell. ISPs or web-hosting service providers often block shell availability to their web, mail, and FTP servers.

Intended as a replacement shell for disabled accounts, nologin refuses a login and displays a default message saying the account is unavailable or disabled. The text file */etc/nologin.txt* contains the customizable, default message.

Complete Exercise 8-9 in Student Workbook

Managing Login Shells and Scripts

12 Authentication and Authorization

As discussed in previous modules, authentication is a mechanism by which a system securely identifies a user. Authorization, however, is a mechanism by which a system determines the level of access an authenticated user has to system resources (e.g., files and directories). Authentication and authorization are tightly coupled. Once a system ensures a user is who they claim to be, it must then prevent unauthorized users from gaining access to secure resources. Authorization is determined by checking file and directory permissions against user access attempts.

12.1 User Accounts

Users needing access to a computer must have a user account for security and accountability purposes. Grant the least privilege necessary for users to perform tasks. System administrators have their own account as well as access to the *root* account for performing administrative functions. The *root* account is unique and cannot be deleted or renamed. Root should be the only account with a UID of 0 and the # sign as its prompt. *Root* user home directory is */root*. The *root* account is the target for many hackers and exploits. Each user has account information in the following files:

/etc/passwd	Used for authentication and authorization
/etc/shadow	Used for authentication
/etc/group	Used for authorization

12.1.1 /etc/passwd File

The */etc/passwd* contains editable user account information for each user. *Root* owns the file and users can read the file. During login, the information provided by the user is compared to their entry in the file and, if it matches, they are authenticated and taken to their home directory. Each line contains fields separated by colons to hold user information. Table 4 details attributes for each user entry.

Table 4. /etc/passwd fields.

1	2	3	4	5	6	7
Username:	Password:	UID:	GID:	Comment:	Home Dir:	Default Shell
`Robert:`	`x:`	`100:`	`11:`	`Office Manager:`	`/home/Robert:`	`/bin/bash`

1. **`Robert`**	User's login name
2. **`x`**	Placeholder for hashed password that is stored in the */etc/shadow* file. Some systems contain the 13 alpha/numeric hashed password
3. **`100`**	User's identification number (UID)
4. **`11`**	Group identification number (GID)
5. **`Office Manager`**	Normally user's comments; full name, office, or phone numbers
6. **`/home/Robert`**	User's home directory
7. **`/bin/bash`**	User's default shell

Information on the *root* account is normally the first entry in the */etc/passwd* file followed by login names used by the system for its operation and system administration. The following lists a few system accounts that exist by default.

- daemon
- bin
- adm
- lp
- sync
- shutdown
- nobody
- apache

System accounts are necessary for specific daemons or services to function properly. To improve system security, the last field of their password account contains the nologin entry. The nologin entry prevents unauthorized user authentication attempts and reduces the risk of an account being exploited to gain access to a valid shell. Additionally, locking down the account and assigning a non-login shell helps to enhance the security posture of the system. Never delete these accounts because their removal could cause Linux to enter a kernel panic.

The syntax for locking down accounts is:

```
[root@CentOS6_A ~]# passwd –l <system account name>
```

Another beneficial security practice is to assign only the read permission to the *other* class of the */etc/passwd* file. This allows the system to access certain information and allows non-root users to view the file contents.

The following lists default permissions for */etc/passwd, /etc/shadow*, and */usr/bin/passwd* files. The two files named passwd, are of different file types and locations. One is located in */etc*, a configuration directory, and the other is located in */usr/bin* a directory contining executable commands. */usr/bin/passwd* is the executable command used to change passwords.

```
[root@CentOS6_A ~]# ls -l  /etc/passwd  /etc/shadow  /usr/bin/passwd
-rw-r--r--.     1  root  root    1702   Nov   4  2015    /etc/passwd
- --- --- ---.  1  root  root     919   Nov  11  2015    /etc/shadow
-rwsr-xr-x.     1  root  root   30768   Feb  22  2012   /usr/bin/passwd
```

12.1.2 /etc/shadow File

For security, only *root* can read and write to */etc/shadow*. Each line in the */etc/shadow* file corresponds to one in */etc/passwd*, making it necessary for user authentication. Lines contain colon-separated fields with user password and aging information. Table 5 illustrates each field.

Table 5. /etc/shadow fields.

1	2	3	4	5	6	7	8
User:	Hashed Password:	Lastch:	Min:	Max:	Warn:	Inactive:	Expire:
bob:	!!$1$C8dw1a0e$...:	11000:	7:	100:	5:	20:	11500:

1.	**bob**	User account name.
2.	**$1$C8dw1a0e$...**	Hashed password.
3.	**11000**	Last password modification date - when last changed.
4.	**7**	Minimum days between password changes
5.	**100**	Maximum days a password is valid before change is required.
6.	**5**	Warning days before a user is required to change their password.
7.	**20**	Number of days after password expires; account is disabled.
8.	**11500**	Number of days from 01JAN1970 until account expires.

If the first character of the hashed password field is not a `$’, Data Encryption Standard (DES) based encryption was used to create the hashed password. In the example above, the first two characters (!!) indicates a locked account. It is followed by the hash field which is comprised of three different fields separated by $’s. The *$1$* represents the cryptographic hasing mechanism used to gernerate the actual hash. The *$C8dw1a0e$* is a randomly generated salt to safeguard against rainbow table attacks, and the last set of characters prior to the “:” is the hash which results from joining the users password with the stored salt and running it rhough the hashing mechanism specified in the first field.

The ‘1’ between the first two `$’s indicate the MD5 (Message Digest 5) algorithm is used with a salt of `C8dw1a0e’. Other values for the password algorithms are: 5 for sha256 and 6 for sha512. The encryption method used is located in */etc/login.defs* and is determined by running the following:

```
[root@CentOS6_A ~]# grep ENCRYPT_METHOD /etc/login.defs
ENCRYPT_METHOD MD5
```

Linux may also use the following additional symbols:

* = A system account that is disabled or never had a password assigned (inactive).

!! = Account locked with the **passwd -l** command or user never logged in.

12.1.3 /etc/group File

The */etc/group* file is where supplemental group membership is maintained. It contains a listing of every group on the system, and user membership in the groups. Table 6 illustrates the fields of */etc/group*.

Operating Systems	The */etc/group* file is where group membership is established and user authorization is managed. Once authenticated, a user must be authorized access to the system and its resources. A user’s primary group is established in the *passwd* file.

Table 6. /etc/group fields

1	2	3	4
Group name:	Password:	GID:	Users having this supplementary group
adm:	x:	4:	**adm,daemon**

1. **adm**	Group name.
2. **x**	Placeholder for the hashed password. Normally not used but useful when implementing privileged groups.
3. **4**	Group Identification (GID).
4. **adm,daemon**	List of users who have this group as a supplementary group.

Example of */etc/group* file contents:

```
root:x:0:
bin:x:1:bin,daemon
daemon::2:bin,daemon
sys:x:3: bin,adm
adm:x:4:adm,daemon
……..
bob:x:501:
```

In the */etc/passwd* file, user *root* has a GID of 0 since root's primary group is the *root* group.

The **groupadd** command creates a new group and the **groupdel** command deletes a group.

Create a new group called *student* with a GID (**-g**) of 411.

```
[root@CentOS6_A ~]# groupadd -g 411 student
```

For a user to create files that can be accessed by members of their supplemental group, use **newgrp** *<groupname>* to switch to the new group before creating the file (similar to using **su** to switch to a user).

If user is currently logged in as a member of the *other* group but would like to create a new file in the *admin* group of which they are also a member:

```
[root@CentOS6_A ~]# newgrp admin
[root@CentOS6_A ~]# vi testfile
```

12.2 User Management

User accounts are created when needed and deleted when no longer required. The only user account installed on the system during installation is *root* user. Management of user accounts requires modification of the */etc/shadow* and */etc/passwd* files. Several commands can be used to make the required changes:

useradd Creates user account or updates default new user information (**-D**).

Options:	**-c**	Comment, an alias for the created account
	-g *<group name>*	Primary group
	-G	Supplementary group
	-d *<home dir>*	Specifies path of user's home directory
	-m	Create a home directory if one does not exist
	-s *<shell path>*	Specifies non-default shell
	-n	Create a user with system's default group value
Note:	If **-n** is not given, a new group with the same name as user is created	
	-D	Displays or change default user values
Note:	The following two options alter the */etc/shadow* file.	
	-e	Specify a login expiration date
	-f	Max days of inactivity before account is invalid

Syntax: **useradd -d /home/***<username>* **-m -n -s** *<shell path> <username>*

```
[root@CentOS6_A ~]# useradd -g Admin -d /home/bill -m bill
[root@CentOS6_A ~]# useradd -d /home/bob -e 09/01/20 –f 30 -c “Inst” -m bob
[root@CentOS6_A ~]# useradd jimmy
```

passwd Updates a user’s authentication tokens; used by root user to set and modify password settings.

Options:

-e *<user>*	Expire password for an account. Force password change.
-i *<num>*	Days before expired password indicates inactive account
-n *<num>*	Sets min field for number of days
-x *<num>*	Sets max field for number of days
-w *<num>*	Sets warning field for number of days
-l *<user>*	Locks user account (-u unlocks a Linux user account)

Syntax: **passwd** *[options] <username>*

```
[root@CentOS6_A ~]# passwd -n 5 -x 90 -w 10 aclee
Adjusting aging data for user aclee
passwd: Success
[root@CentOS6_A ~]# passwd -e jill
Expiring password for user jill.
passwd: Success
```

chage Change and view user password expiration (age) information.

Options: **-l** Displays account aging information

```
[root@CentOS6_A ~]# chage -l jill
Last password change                                  : Apr 04, 2016
Password expires                                      : never
Password inactive                                     : never
Account expires                                       : never
Minimum number of days between password change        : 5
Maximum number of days between password change        : 90
Number of days of warning before password expires     : 10
[root@CentOS6_A ~]#
```

usermod Change some of the parameters set with **useradd**.

Options:
-s *<login shell>*
-g *<primary group>*
-G *<supplementary group>*
-a *<allow more than one supplementary group>*

Syntax: **usermod** *[options] <username>*

```
[root@CentOS6_A ~]# usermod -s /usr/bin/bash  Robert
[root@CentOS6_A ~]# usermod -g admins Robert
[root@CentOS6_A ~]# usermod -G students Robert
```

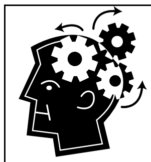
`userdel`	Delete a user account.		
	Options:	`-r` *<username>*	Recursively removes user home directory
		`-f`	Force removal even if logged in

```
[root@CentOS6_A ~]# userdel -rf Robert
```

The **`userdel`** command removes all entries in */etc/passwd*, */etc/group*, and */etc/shadow* files. It does not remove user's home directory unless the **-r** option is used.

pwconv Convert information in */etc/passwd* to */etc/shadow* and report any errors in */etc/passwd*. After modifying */etc/passwd*, **pwconv** makes the necessary changes by creating a new line in */etc/shadow* that corresponds to the new user.

`id` Print real and effective (UID) and (GID).



Complete Exercise 8-10 in Student Workbook

Creating and Managing User and Group Accounts

Objectives

Day 5

At the end of this training day, students will:

- Control access to a Linux network using service configuration files.
- Identify files used in determining network ports and their mapped services.
- View and modify network interface settings.
- Configure file sharing in Linux using NFS and effective commands.

Exercises

This training day includes the following exercises:

- Exercise 8-11, Xinetd in Linux
- Exercise 8-12, Sharing with Linux NFS Server

JCAC-eTC

Complete self-study activities and assignments in the online training environment.

13 Networking Services Administration

Most network services in Linux are daemons. They run in the background and support network-based applications when needed. These may include connecting to the internet, performing file transfers from one host to another, or any other type of remote access. This section introduces configuration files that control access to available network services by modifying their parameters.

13.1 Linux Network Configuration

Historically, an Internet daemon, *inetd*, provided Internet service management for Unix systems using System V. *Inetd* reduced system load by starting other daemons only when needed and provided simple Internet services internally without invoking other daemons. *Inetd* listened to multiple ports, invoked a requested service, and started at boot with configuration information in */etc/inetd.conf*. This file set the basic parameters for each service that *inetd* controlled. Unless the service line in *inetd.conf* was commented out, the service started when read by *inetd*. Services were simply on or off with no other form of security.

Linux Version 6 and other systems using Upstart or Systemd extended the Internet service management by implementing the *xinetd* daemon. *Xinetd* provides the same function as *inetd* and also provides enhanced secure network service capabilities. *Xinetd* has a configuration file, */etc/xinetd.conf*, that sets the basic parameters for all services it manages. Some of these parameters are listed below:

- Access control capabilities
- Time-based access
- Full logging for connection success or failure
- Limitation on total number of servers
- Limitation on size of the log files

After executing all parameters in *xinetd.conf*, all configuration files located in the */etc/xinetd.d* directory are read. The */etc/xinetd.d* directory contains a configuration file for each *xinetd* managed service. Each *xinetd* service configuration file contains a *disable* parameter which, by default, is set to *yes*. To enable the service, edit the configuration file and set disable to *no*. In order for changes to take affect, restart *xinetd*. Changes take affect immediately without interrupting other services.

```
[root@CentOS6_A ~]# /etc/rc.d/init.d/xinetd restart
```
…or…
```
[root@CentOS6_A ~]# service xinetd restart
```

Entries in the configuration file consist of a block of attributes defined for different features. These can be server program name, protocol, and security restrictions. Each block in the entry is preceded by the keyword `service` and the service name. The block of attributes is enclosed with braces {}. Each attribute begins with the attribute name, followed by an operand (e.g., =, =+, or =-) and the value. Some attributes are required. Others may be included to help provide security.

Below is an example of the Telnet configuration file */etc/xinetd.d/telnet* indicating the service is enabled.

```
service telnet
{
flags                  =  REUSE
socket_type            =  stream
wait                   =  no
user                   =  root
server                 =  /usr/sbin/in.telnetd
log_on_failure         += USERID
disable                =  no
}
```

Most attributes typically support only one operator (=) that sets that value. Some attributes support multiple operators, like *log_on_failure* above. By adding a +, the *log_on_failure* example adds the values defined in the block to the values previously defined for *log_on_success* in the default statement in */etc/xinetd.conf*.

Other entries may also be defined, including servers that are activated when requested, along with any options and security precautions.

The following example extracted from */etc/xinetd.d/ftp* only allows the ftp service to be used by 192.168.1.0/24 and only between 0900 and 1500 hours. This is an example of a service with restrictions applied for security reasons.

```
service ftp
{
socket_type            = stream
wait                   = no
user                   = root
server                 = /usr/sbin/in.ftpd
only_from              = 192.168.1.0/24 #, only for internal use
access_times           = 09:00-15:00
disable                = no
}
```

The following example, extracted from */etc/xinetd.d/pop3*, sends a failed logon attempt to the local host if the service is enabled. The pop3 service is currently disabled, so that attribute setting is ignored.

```
service pop3
{
socket_type          = stream
wait                 = no
user                 = root
server               = /usr/sbin/ipop3d
log_on_failure       += HOST
disable              = yes
}
```

Some optional attributes of network services are listed in Table 7.

Table 7. Services attributes.

Attribute	**Values and Description**
log_type	xinetd uses syslogd and *daemon.info* selector by default. • SYSLOG selector [level]: allows to choose among daemon, auth, user or local0-7 from syslogd; • FILE [max_size [absolute_max_size]]; specified file receives information. Two options set file size limit.
log_on_success	Different information can be logged when a server starts: • PID: server's PID (if it's an internal xinetd service, PID has a value of 0); • HOST: client address; • USERID: identity of remote user • EXIT: the process exit status; • DURATION: session duration.
log_on_failure	• HOST, USERID: like above mentioned; • ATTEMPT: logs an access attempt. • RECORD: logs all information available on the client.
no_access	List of clients not having access to this service.
access_times	Specifies time range when a particular service may be used. Time range must be stated in 24-hour format notation, HH:MM-HH:MM.
only_from	List of authorized clients. If attribute has no value, service access is denied.
Port	Port associated to the service.
Protocol	Specified protocol must exist in */etc/protocols* file
Server	Path to server.
server_args	Arguments to be given to server.

socket_type	stream (TCP), dgram (UDP), raw (IP direct access) or seqpacket ().
Type	xinetd can manage three types of services: 1. RPC: for those defined in */etc/rpc* file … but doesn't work very well; 2. INTERNAL: for services directly managed by xinetd 3. UNLISTED: services not defined in */etc/rpc* file or */etc/services* file
Wait	Defines service behavior toward threads. Two values are acceptable: • yes: service is mono-thread, only one connection can be managed by the service; • no: a new server is started by xinetd for each new service request

13.2 Starting and Stopping Network Services

There are two ways of invoking network services in CentOS 6; edit the service file in */etc/xinetd.d* or use **chkconfig**. For example, to start *telnet*, edit */etc/xinetd.d/telnet* and change the disable attribute from *yes* to *no*. Save the file. A Telnet example file is shown below.

```
service telnet
{
flags                 = REUSE
socket_type           = stream
 wait                 = no
 user                 = root
 server               = /usr/sbin/in.telnetd
log_on_failure        += UID
disable               = no
}
```

In CentOS 6 **chkconfig** controls network services located in *xinetd.d* and non-network services in rc run-levels. If *xinetd* is running, network services are immediately updated without restarting the daemon. However, changes to network service attributes do require a restart.

Syntax: **chkconfig** *<service>* **on**

Example: **chkconfig telnet on**

The command **chkconfig --list**, lists all services it knows about, and whether they are stopped or started in each run-level.

To list only the Telnet service status use **chkconfig --list** followed by the service name.

```
[root@CentOS6_A ~]# chkconfig --list telnet
telnet:                   on
```

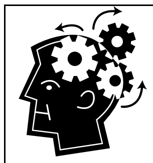
CentOS 7 uses systemd and the **systemctl** command. The service must be enabled prior to being started. Note in CentOS 7, the Telnet service is named *telnet.socket*.

Syntax: **systemctl** *<option> <service>*

Example: **systemctl enable telnet.socket**
systemctl start telnet.socket

An example of listing only the Telnet service status:

```
[root@CentOS7_A ~]# systemctl status telnet.socket
telnet.socket - Telnet Server Activation Socket
Loaded: loaded (/usr/lib/system/system/telnet.socket; enabled; ...
Active: inactive (dead) since Mon 2019-07-15 09:11:30 CDT; ...
Docs: man:telnetd(8)
<truncated>
```



Complete Exercise 8-11 in Student Workbook

Xinetd in Linux

13.3 Network Service Mapping

xinetd.conf is the configuration file that determines the service provided by *xinetd*. The */etc/services* file lists service-to-port mappings. Modify */etc/services* to add newly created services, but ensure those services point to the configuration file.

The following is a truncated excerpt from a CentOS network service file showing the format and fields.

```
[root@CentOS6_A ~]# more /etc/services
# /etc/services:
# $ID: services, v 1.48 2009/11/11 14:32:31 ovasik Exp $
#
# Network services, Internet style
#
#The well known ports are those from 0 through 1023.
#The Registered ports are those from 1024 through 49151
# The Dynamic and /or Private Ports are those from 49152 through 65535
#
# Each line describes one service, and is of the form:
#
# service-name  port/protocol  [aliases ...]   [# comment]
ftp             21/tcp
ssh             22/tcp                          # Secure Shell
telnet          23/tcp
smtp            25/tcp          mail
sunrpc          111/tcp         rpcbind         # Portmapper
ldap            389/tcp                         # Lightweight Directory
# Host specific functions
tftp            69/udp
finger          79/tcp
netbios-ns      137/tcp                         # NETBIOS Name Service
netbios-dgm     138/tcp                         # NETBIOS Datagram Service
netbios-ssn     139/tcp                         # NETBIOS Session Service
netbios-ssn     139/udp                         # NETBIOS Session Service
# UNIX specific services
login           513/tcp
```

```
shell           514/tcp         cmd             # no passwords used
who             513/udp         whod
syslog          514/udp         rsyslog
nfs            2049/udp         nfs             # NFS server daemon (clts)
nfs            2049/tcp         nfs             # NFS server daemon (cots)
lockd           4045/udp                        # NFS lock daemon/manager
```

Remote Procedure Call (RPC) defines a system-independent way for processes to communicate over a network and provides for IPC, allowing a program running on one computer to execute code on a remote system. On a Linux system, rpcbind is mapped to TCP port 111. Knowing well-known ports and service mappings is a good enumeration technique for determining an OS. RPC services are configured in */etc/rpc* in a similar manner as */etc/services*.

The */etc/rpc* file maps RPC services defined in *xinetd.conf* to the appropriate RPC program numbers. Linux uses the alias *portmap*, when referring to rpcbind. When an RPC server starts up, it registers with rpcbind listing the services it supports and ports on which it can be contacted. Clients query rpcbind to find out how to get in touch with an appropriate server.

The format and column descriptions for the contents of */etc/rpc* follows:

Program name	Program number	Description
rpcbind	100000	portmap sunrpc rpcbind

To see the contents, **more /etc/rpc**:

```
[root@CentOS6_A ~]# more /etc/rpc
#
#ident      "@(#)rpc    1.17  01/10/30 SMI"     /* SVr4.0 1.2     */
#
#
#     rpc
#
portmapper        100000      portmap sunrpc rpcbind
rstatd            100001      rstat rup perfmeter
rusersd           100002      rusers
nfs               100003      nfsprog
ypserv            100004      ypprog
mountd            100005      mount showmount
ypbind            100007
walld             100008      rwall shutdown
yppasswdd         100009      yppasswd
etherstatd        100010      etherstat
rquotad           100011      rquotaprog quota rquota
<output truncated>
```

The **rpcinfo –p** command displays a list of RPC services mapped to their corresponding port.

```
[root@CentOS6_A ~]# rpcinfo -p | more
   program vers proto   port  service
    100000    4   tcp    111  portmapper
    100000    3   tcp    111  portmapper
    100000    2   tcp    111  portmapper
    100000    4   udp    111  portmapper
    100000    3   udp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp  35036  status
    100024    1   tcp  37836  status
```

13.4 Configuring an IP Address

Linux uses the interface configuration command, **`ifconfig`**, to view and modify an IP. The **`-a`** option displays the status of all interfaces, even those that are down.

```
[root@CentOS6_A ~]# ifconfig -a
eth4      Link encap:Ethernet  HWaddr 00:0C:29:22:1D:C9
          inet addr:192.168.0.80  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe22:1dc9/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:5800 errors:0 dropped:0 overruns:0 frame:0
          TX packets:999 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:580491 (566.8 KiB)  TX bytes:118425 (115.6 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:46 errors:0 dropped:0 overruns:0 frame:0
          TX packets:46 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3132 (3.0 KiB)  TX bytes:3132 (3.0 KiB)
[root@CentOS6_A ~]#
```

Common interfaces are:

lo	Loopback interface
eth0	Ethernet interface. (Additional interfaces are eth1, eth2, etc.)
pcn0	Ethernet (standard “VM” interface)
wlan0	Wireless network interface.

In the above example, eth4 indicates the 4th ethernet interface. Throughout this module, an “*X*” trailing the interface such as eth(*X*) indicates the valid Ethernet for that scenario. This applies to tests and quizzes, as well.

Syntax for manually setting an IP address:

`# ifconfig –a`	Display status of all network interfaces, active or inactive.
`# ifconfig eth0`	Used to view network settings on first Ethernet adapter installed.
`# ifconfig eth0 down`	Turns off interface eth0; it does not send or receive.
`# ifconfig eth0 up`	Turns on interface eth0; allowing it to send and receive.
`# ifconfig eth0 192.168.1.102 netmask 255.255.255.0 broadcast 192.168.1.255`	Assigns eth0 an IP, netmask, and broadcast address.

Interface configuration files are located in */etc/sysconfig/network-scripts*. The directory contains one or more files, each representing an installed NIC (e.g., ifcfg-eth0, ifcfg-eth1, ifcfg-wlan0). Administrators use these files to individually control how each interface functions.

Another key networking file is */etc/hosts*. This text file maps IP addresses to hostnames, one line per IP address. It acts as a resolver for machines on a local network, bypassing Domain Name System (DNS) resolution. This speeds up internal communications.

14 Network File System (NFS)

NFS is a native Linux distributed file system protocol allowing a server to share directories and files with clients over a network. It allows file system sharing among networked computers, disks, or devices like CD-ROM drives. NFS provides a way to mount more file systems into a local file system hierarchy. It includes a mounting protocol, a mount server, daemons that coordinate file service, and several diagnostic utilities.

NFS runs on top of the RPC protocol defined earlier. Although NFS originally used UDP as a transport protocol, it lacked any kind of flow/congestion control necessary for efficient performance. The remedy was to use TCP as the transport protocol. However, servers supporting TCP also accept connections from a UDP client. The NFS client chooses the transport protocol.

NFS server setup involves the addition of the portmap and NFS-utilities packages, resulting in three additional components:

Portmap	Maps calls made from other machines to the correct RPC service.
NFS	Translates remote file sharing requests into local file system requests.
rpc.mountd	Daemon that mounts and unmounts (**`umount`**) file systems.

Creating a directory to share with NFS is similar to creating a shared directory in Windows. The first step is to enter share information in the appropriate server configuration file followed by a command to make the share take effect. NFS refers to shares as exports, therefore, the */etc/exports* file contains a table of local physical file systems on an NFS server that are accessible to NFS clients. After saving the configuration information, the command **`/usr/sbin/exportfs -a`** updates the current table of exports for the NFS server.

To set up an NFS share between two Linux systems both the server and client require configuration.

Server Setup:

Step 1: Create a share in */etc/exports*.

Exports contain specific access controls. Each line contains an export point, client(s) allowed to mount the file system, and a list of export options.

Options:		
	rw	Exported file system is mounted read/write
	ro	Exported file system is mounted read only
	[ ]	If an empty space exists after host/IP, read only (ro) is the default, as shown in the second example.

Format:

```
<share_directory> <hostname_or_IP>(<permissions>)
<share_directory> <hostname1>(<options) <hostname2>(<options>) …
```

Example entries:

```
/myshare *.local.domain(rw)
/myshare 192.168.0.82 *(rw)
```

Step 2: Ensure NFS service is running. The **`service`** command stops, starts, or displays the status of the *nfs* script that controls the daemons needed for file sharing.

```
[root@CentOS6_A ~]# service nfs {status  | stop  | start}
```

Step 3: Share all file systems identified in */etc/exports*:

```
[root@CentOS6_A ~]# exportfs -a
```

Client Setup:

Step1: Ensure services needed for NFS are running.

Step 2: Search for a machine with an exported file system.

```
[root@CentOS6_A ~]# showmount -e  <IP Address>
```

Step 3: Mount the file system.

```
[root@CentOS6_A ~]# mount <hostname or IP>:<path> <mountpoint>
```

The **`umount`** command removes a mount:

```
[root@CentOS6_A ~]# umount /mnt/linux.share
```

Complete Exercise 8-12 in Student Workbook

Sharing with Linux NFS Server

Objectives

Day 6

At the end of this training day, students will:

- Assess host and network security, configurations, and vulnerabilities.
- Explain LDAP features in relation to BIND.
- Identify SAMBA server configuration elements for interoperability with Windows.
- Identify Apache configuration file (httpd.conf) components.
- Modify Apache directive values to limit access to server directories and files.
- Use the Telnet protocol and HTTP methods to gain Apache web server information.
- Recall patch procedures and determine patch signatures and version of a target system.

Exercises

This training day includes the following exercises:

- Exercise 8-13, Samba Sharing with Linux and Windows
- Exercise 8-14, Configure Apache and Setup Host-based Access Controls
- Exercise 8-15, Identify OS Version and Patch Level

JCAC-eTC

Complete self-study activities and assignments in the online training environment.

15 Name Resolution

Name resolution is the use of a service to translate human-friendly hostnames into IP addresses. This name resolution occurs locally and over a network. The OS has multiple options available to accomplish name resolution within a computer network.

15.1 Controlling Hostname Lookup

All Linux name services using DNS, LDAP, or NIS/NIS+ have a Name Service Switch (NSS) file, */etc/nsswitch.conf*, configured. */etc/nsswitch.conf* is used by GNU C Library functions to determine which name service should be used to gather information from various categories, and in what order. Each category of information is identified using a file or database name that GNU C recognizes, i.e., */etc/hosts, /etc/shadow*, */etc/ethers*, */etc/group*, */etc/passwd,* or **.db* files.

The configuration file has a line entry for each service. The first field contains a database name followed by a colon. The second field contains a list of possible source databases.
Below is an excerpt from */etc/nsswitch.conf* configured to use DNS.

```
[root@CentOS6_A ~]# more /etc/nsswitch.conf
# An example Name Service Switch config file.  This file should be
# sorted with the most-used services at the beginning.
# <truncated>
# Valid entries include:
#
#  nisplus     Use NIS+ (NIS version 3)
#  nis         Use NIS (NIS version 2)
#  dns         Use DNS (Domain Name Service)
#  files       Use the local files
#  db          Use the local database (.db) files
<truncate>
# To use db, put the “db” in front of “files” for entries you want to be
# looked up first in the database
#
# Example:
#passwd:       db files nisplus nis
<truncate>
passwd:        files
group:         files
hosts:         files dns
<Output truncated>
```

The order of services listed determines the order NSS attempts to use those services to resolve queries on the specified database.

Name service starts by looking at settings next to the hosts’ entry in */etc/nsswitch.conf*. It reads local files first (e.g., */etc/hosts, /etc/shadow,* and */etc/passwd*). If it is necessary to continue, each name service is read from left to right, stopping only if the IP is found or if the name service list is complete.

Although not widely used, NIS+ uses a hierarchical structure; multiple servers in a domain can run NIS+. The primary server is the master and backup servers are replica servers. NIS+ table permissions determine a user's level of access to table contents.

15.2 Local Name Resolution

Configure local name resolution in the */etc/hosts* file using the hosts IP address and FQDN followed by any aliases.

Example of */etc/hosts*:

```
[root@CentOS6_A ~]# more /etc/hosts
127.0.0.1 localhost localhost.jcac.local localhost.local localhost loghost
::1       localhost localhost.jcac.local localhost.local localhost loghost
192.168.0.80    CentOS6_A.jcac.local    CentOS6_A
192.168.0.81    CentOS6_B.jcac.local    CentOS6_B
192.168.0.82    Apache.jcac.local       Apache
192.168.0.83    CentOS7_A.jcac.local    CentOS7_A
192.168.0.51    WinXP.jcac.local        WinXP
```

Referencing a hostname, FQDN, or an alias has the same effect as using any of them. For example, **`telnet 192.168.0.82, telnet apache.jcac.local`**, and **`telnet apache`** provide the same results. Configuration of this file is convenient, but requires an update to the */etc/hosts* file on every system on the local network.

Creating a DNS server allows clients on the local network to communicate with systems locally and in a domain more efficiently, especially in larger network environments.

15.3 Berkeley Internet Name Domain (BIND)

Berkeley Internet Name Domain (BIND) is the most widely used DNS software on the Internet for domain name resolution. It implements the *named* service, a DNS server, to provide name resolution services or act as an authority for a domain or subdomain. The named service executes the DNS server daemon. BIND's main configuration file, */etc/named.conf*, specifies the names and locations of root servers and zone files. BIND contains a domain name resolver, an authoritative DNS server, and several diagnostic and operational tools.

The resolver is a set of C library routines that sends name queries to appropriate servers, and then responds to the servers' replies. */etc/resolv.conf* contains information read by the resolver routines. Some applications, such as the Apache web browser, also use a local stub resolver library on the same computer to look up names. The stub resolver is a part of the OS and forwards queries to a caching resolver, server, or group of servers dedicated to DNS services. These resolvers send queries to one or more authoritative servers.

The authoritative DNS server replies to requests from resolvers using information from its database. The database contains the zones the server is authoritative for and IP address to name mappings for each system within the zone.

DNS clients use */etc/resolv.conf* to determine the location of a DNS server and the domain to which they belong. Entries in */etc/resolv.conf* contain a keyword, value, and a description. Table 8 provides a breakdown of the fields contained in */etc/resolv.conf*.

Table 8. Resolv.conf keywords.

Keyword	Value	Description
Search	jcac.com	Helps nameserver get information quickly. It compares the entry to forward and reverse zones.
Nameserver	10.90.1.254	IP address of a nameserver the resolver should query.
Domain	192.168.0.81	Local domain used by default.

Example of */etc/resolv.conf*:

```
[root@CentOS6_A ~]# more /etc/resolv.conf
# Name server(s) and IP
search  jcac.com
nameserver  10.90.1.254
nameserver  8.8.8.8
```

Table 9 lists some of the files needed to configure DNS and their function. These files control how a system works at hostname to IP address resolution. Network services that rely on name resolution are affected by configuration (or misconfiguration) of these files.

Table 9. DNS associated files.

File location	Function or purpose	Restart *named*?
/etc/hosts	Map of IP addresses to hostnames and aliases	No
/etc/nsswitch.conf	Controls how hostnames are resolved system-wide	No
/etc/resolv.conf	Specifies machines to use for name resolution	No
/etc/named.conf	Main configuration file for BIND	Yes

15.4 Lightweight Directory Access Protocol (LDAP)

LDAP is an Internet protocol used by email and other programs to look up information from a server. LDAP is just one of many protocols that Linux systems use for accessing directory-like information, where fast lookups and less-frequent updates are normal.

Windows	LDAP is an open standard protocol for accessing object-oriented databases, known as Directory Servers. LDAP defines how clients access the directory server and perform database operations over an IP network.

16 Samba

Samba is a suite of software developed to provide file and print sharing between different OSs. Interoperability between Linux, Windows, and MAC based clients is nearly impossible without the use of specialized software like Samba. Samba uses the SMB protocol to provide secure, stable, and fast file and print services for all clients. Windows clients, use SMB, also known as CIFS, on TCP 445, and NetBIOS, on TCP 139, as file and print sharing protocols. Samba mimics SMB/CIFS and NetBIOS on Linux hosts, opening those same ports, to allow share negotiations with Windows systems.

Samba provides five basic services.

- File sharing
- Network printing
- Authentication and authorization
- Name resolution
- Service announcement - file server and printer browsing

Samba's functionality comes from two daemons: *smbd* and *nmbd*.

smbd	Implements file and print services, as well as authentication and authorization.
nmbd	Provides other major SMB/CIFS components: name resolution and service announcement.

The daemons can be stopped, started, or have their status displayed using the `service` command without the trailing `d'.

Example: `service smb [argument]` or `service nmb [`*`argument`*`].`

To execute the daemons:

`/etc/init.d/smb restart` and `/etc/init.d/nmb restart`.

Running Samba requires no kernel modifications and is entirely a user process. It binds to sockets used for NetBIOS and waits for a client to request access to a resource. Once the request is accepted and authenticated, *smbd* forks an instance of itself that runs as the user making requests.

Samba's configuration file, */etc/samba/smb.conf*, is read by the *smb* service. This file specifies all shared directories and printers along with access rights and general parameters. It documents all options for integrating into a network with Microsoft machines. Use the `testparm` command to check *smb.conf* for internal correctness after making changes.

Changes made to *smb.conf* are normally instantaneous since the daemons, *smbd* and *nmbd*, are constantly running in the background. However, to be sure changes take effect immediately, run the `service` command to force the daemons to evaluate the configuration file.

16.1 Samba Configuration Sections

The two key sections of *smb.conf* are *global settings* and *share definitions*.

The *global settings* section defines server-wide settings such as *Network-Related Options*. These include workgroup name, local interface information, and host access controls. Global settings apply default options to shares not already defined.

The s*hare definitions* section applies to directories, files, and printers. The key share sections are *[global]*, *[homes*] and *[printers]*. The names inside the square brackets delineate the unique sections. Each section names the share, or service, to which the section refers. For instance, the *[global]* section contains settings affecting the overall environment. The *[homes]* section is a unique disk share containing options that map to a specific directory on the Samba server. The *[printers]* share contains options that map to various printers on the server.

All sections, except for *[global]*, are available as a disk or printer share to clients connecting to the Samba server. The lines within each section are individual configuration options for that share. Each option is assigned a value in the format: `option = value`.

Capitalization is not important and comments in the configuration file start with either a hash (#) or a semicolon (;).

smb.conf section configurations:

```
[global]
   workgroup = JCAC
   security = user
   encrypt passwords = yes

[homes]
   browsable = yes
   map archive = yes
   read only = yes
   path = /usr/local/samba/tmp

[printers]
   path = /var/tmp
   printable = yes
   min print space = 2000
   guest ok = yes
```

16.2 Samba Security Modes

Security modes are set in the Samba configuration file under the *global* section of the *share definitions*. These modes affect how clients respond to Samba and are the most important settings in the *smb.conf* file. Listed below are explanations of different security modes.

User	(Default) Written as *security = user*, it is the most common setting used for a standalone file server or a domain controller. In user level security, a client must logon with a valid username/password.
Domain	Written as *security = domain*, Samba tries to validate the username/password by passing it to a Windows domain controller, in the same way that a Windows server does. The Samba server has a domain security trust account and all authentication requests are passed to the domain controller.
Active Directory	Written as *security = ads*, Samba acts as a domain member in an ADS realm. To operate in this mode, the machine running Samba must have Kerberos installed and configured, and Samba must be joined to the ADS realm as an AD member.

16.3 Samba Share Creation

Procedures for creating a Samba share include adding users with proper credentials to the *pdbedit* file, creating a share, setting the share's security context, and manipulating the Samba configuration file, as explained below:

Step 1: Ensure Samba is running by checking the status of smb and nmb services.

```
[root@CentOS6_A ~]# service smb {restart | stop | start | status}
[root@CentOS6_A ~]# service nmb {restart | stop | start | status}
```

Step 2: The **pdbedit** command is used to manage Samba user accounts located in a private password database (*passdb.tdb*). **pdbedit** is a *root* user command used to manage account security and policy settings.

```
[root@CentOS6_A ~]# pdbedit -a  student
New SMB password: <new password>
Retype new SMB password: <retype new password>
Added user student
```

Step 3: Create share and set security context.

On systems running SELinux, all processes and files are labeled in a way that represents security relevant information. This information is the SELinux security context. This context, viewed with `ls –Z`, is a string of three or four words separated by a colon.

Each word represents a component of the security context: user, role, type, and level of classification of the file or process. The **ls –Z** command provides the security context of the */home/share* directory. The result includes the labels provided by SELinux to make access control decisions:

```
[root@CentOS6_A ~]# mkdir  /home/share
[root@CentOS6_A ~]# ls -Z  /home
drwxr-xr-x. root root unconfined_u:object_r:home_root_t:s0 share
```

- User (*unconfined_u*)
- Role (*object_r*),
- Type (*home_user_t*)
- Level (*s0*)

By default, the context *type* is normally that of the user's home directory, i.e., *home_user_t*. To permit users to write to the Samba share, change the context type to read *samba_share_t*.

The **chcon** command changes the security context type of a file or process. The **-R** option is used to recursively affect subdirectories and all files.

```
[root@CentOS6_A ~]# chcon -t  samba_share_t  -R  /home/share
[root@CentOS6_A ~]# ls -Z
drwxr-xr-x. root root unconfined_u:object_r:samba_share_t:s0 share
```

Step 4: Verify/change ownership of the share.

Use the **chown** command to modify the ownership of the /home/share directory. Use a colon to delimit owner and group when changing both.

```
[root@CentOS6_A ~]# chown -R student /home/share
[root@CentOS6_A ~]# ls -Z /home
drwxr-xr-x. student root unconfined_u:object_r:samba_share_t:s0 share
```

Step 5: Modify the Samba configuration file sections as discussed previously.

Complete Exercise 8-13 in Student Workbook

Samba Sharing with Linux and Windows

17 Apache Web Server

Apache is the most widely used web server software due to its availibility for many OSs, including Windows. The Linux configuration file is */etc/httpd/conf/httpd.conf*. This file contains configuration information for access capabilities, file restrictions, and logging.

17.1 Apache Configuration

Directories and files associated with Apache are stored in different locations depending upon the version of Linux. The *httpd.conf* file contains directives grouped into three basic sections. Each section has a set of directives or instructions related to directories, files, and rules affecting that section.

Section 1 - Global Environment	Contains directives that control the global operation of the Apache server process.
Section 2 - Main server	Contains directives that define the parameters of the main server, which responds to requests not handled by a virtual host.
Section 3 - Virtual Hosts	Contains settings for a virtual host, if one exists. Identifies virtual hosts and defines some of their directives. Container used to maintain multiple domains/hostnames on one server.

The Apache configuration file contains only one directive per line. Directives are not case sensitive; however, arguments are case sensitive.

To check the file for syntax errors without restarting the server:

```
[root@Apache ~]# apachectl -t
Syntax OK
```

To restart the httpd service after configuration changes:

```
[root@Apache ~]# service httpd restart
```

17.2 Global Environment Directives

Many directives are contained in the global environment section. The following directives identify common configurations and describe ways to neutralize vulnerabilities associated with web-based applications.

ServerTokens

Works with the *ServerSignature* directive located in the *Main* server section. By default, the server sends an HTTP response header with server version information to the requesting source. If not hidden, a hacker may attempt to exploit vulnerabilities in unpatched or older versions of Apache. This technique is known as banner grabbing and is discussed later.

If *ServerSignature* is *on*, *ServerTokens* can be changed from *Full/OS*, revealing version information, to *Prod* (Product Only), revealing "Apache" and no version information. If *ServerSignature* is *off*, it hides Apache version information on any error pages when accessed from most web browsers.

```
[root@Apache ~]# more httpd.conf
#ServerTokens Prod
<output truncated>
ServerSignature On
```

ServerRoot

Defines the top of the directory tree where the server's configuration, error, and log files are stored. For the version of Linux used here, all directories and files related to Apache are stored under */etc/httpd*.

```
[root@Apache ~]# more httpd.conf
ServerRoot "/etc/httpd"
<output truncated>
```

Listen

Instructs the server to accept incoming requests on the specified port or IP address-and-port combination. When only given a port number, the server listens to the given port on ALL interfaces. If given an IP address-and-port combination, the server listens on the given port and interface and responds to requests received. More than one listen directive may be used.

```
[root@Apache ~]# more httpd.conf
Listen 80
Listen 192.168.0.82:80
```

User/Group

Sets the User and Group under which the server answers requests. User *apache* and group *apache* run *httpd* by default. If user is *root* or *nobody*, the web server may be vulnerable to exploitation.

```
[root@Apache ~]# more httpd.conf
User  apache
Group  apache
```

17.3 Main Server Directives

The *Main Server* directives define parameters of the main or default server, which responds to requests not handled by a virtual host. These directives also provide default values for the settings on all virtual hosts. There are many directives and options available in this section, a few are listed below.

ServerAdmin	Identifies the email account of where to send server problems. ServerAdmin root@localhost
ServerName	Provides the name and port the server uses to identify itself. Uses the IP address if no registered DNS name exists. ServerName 192.168.0.82
DocumentRoot	Default directory that contains all web page documents. DocumentRoot “/var/www/html”
Options	Consist of up to five parameters that control access to the server’s file system. This directive applies to directory containers that contain more than a single line of information.

Indexes	By default, a web request looks for *index.html*. If not found, an error is sent to the requester. If Index is set, the web server provides a directory listing to help a user navigate and search for the file.
Include	Provides dynamic content to an HTML page. Dynamic content is information that may change quickly, such as updates to a forum.
FollowSymLinks	Allows a web user to follow a link to another directory. Take caution in case a link points to an unsecured portion of the file system.
ExecCGI	Allows execution of CGI scripts for the directory.
MultiViews	A per directory option that can be set in *httpd.conf* or in *.htaccess*.

Possible values for the Options directive are “None", "All", or any combination of the above. However, MultiViews must be named explicitly, "Options All" does not provide.

17.3.1 Container Directive

The directories Apache has access to are configured with respect to what services and features are allowed and/or disabled in that directory and its subdirectories. The first line of the container directive begins with a line similar to the example below.

<Directory “/var/www/html”>

The next line(s) of the container directive identify *Option* directives.

Options Indexes FollowSymLinks
AllowOverride None - Defines which access directive can be overwritten by *.htaccess*

The next few lines control access to the directory. The first of these lines, *Order*, tells what order to evaluate the *allow* and *deny* lines. The example below allows from all first, and then deny from 192.168.0.81 specifically.

Order allow,deny
Allow from all
Deny from 192.168.0.81

The final line closes the container directive: </Directory>

17.3.2 Apache Log Directives

The entries below are configured to enable an administrator to manage the amount and type of messages that are recorded in the logs.

ErrorLog	Location of the error log file. Logs are located in */etc/httpd/logs*. The file that stores error log information is */etc/httpd/logs/error_log*. There is also a hardlink to log files in the */var/log/httpd* directory.
LogLevel	Controls the types of messages logged to the *error_log*. Log levels from lowest to highest include: debug, info, notice, warn, error, crit, alert, and emerg. Errors for the set level and any levels above that, reside here. For example, “warn” reports warn, error, crit, alert, and emerg levels. Normally, message levels of “notice” cannot be suppressed. This appears when *httpd* starts or restarts.
LogLevel warn	The *error_log* displays a line of text for every error identified by the LogLevel. Information includes client DTG, log level [warn], and information related to the client’s request.

```
[root@Apache ~]# less  /etc/httpd/logs/error_log
[Thu Sep 03 11:55:03 2015] [notice] [client 192.168.0.80] Directory
index forbidden by Opt…
```

CustomLog	Included in this directive are access, agent, and referrer information in a combined logfile format. The file that stores a record of access attempts to the web is */etc/httpd/logs/access_log*.

The *access_log* displays a line of text for every access attempt. The client's IP, DTG, HTTP Method (GET, POST), header information, HTTP Error Response, web browser and OS information is displayed. An HTTP method indicates what to perform on the object identified by the URL. A few methods are GET, HEAD (GET without HTML body), PUT, POST, and LINK.

```
[root@Apache ~]# tail /etc/httpd/logs/access_log
192.168.0.82 – - [03/Sep/2015:12:00:00 -500] "GET / HTTP/1.1" 200 2326
"http//10.90.111.8…
```

17.4 Virtual Hosts

The *<VirtualHost>* section allows multiple web sites to run on a single server. Virtual hosts can be name-based (multiple names run on each IP) or IP-based (a different IP exists for each web site. If a virtual host exists, directives in a *<VirtualHost>* container override directives in the *Main* server section.

17.5 HTTP Request/Response Headers

When a browser requests a web page, the web server receives an HTTP Request Header. The web server replies with a HTTP Response Header. This is how a web server identifies itself to incoming requests, but this may pose security risks. Attackers can use the HTTP Request/Response Header as a first step to exploit an unpatched or outdated system.

`Telnet` can act as a tool to gain information from a web server. When a client uses **`telnet`** to access a server's IP address on Port 80 (HTTP), a **HEAD** request is sent and the server returns only address *header* information. In addition to getting header information from the server, sending a **GET** request returns the entire header and message body. To prevent this, turn signature *off* within the *httpd.conf* file. See the following example:

Telnet to the Apache webserver's HTTP port. When connected, enter **`HEAD / HTTP/1.0`** to reveal web server version and OS information.

```
[root@CentOS6_B ~]# telnet 192.168.0.82 80
Trying 192.168.0.82…
Connected to 192.168.0.82
Escape character is '^]'
HEAD / HTTP/1.0

HTTP/1.1 403 Forbidden
Date: Wed, 11 Dec 2019 16:51:14 GMT
Server: Apache/2.2.15 (CentOS)
```

Repeat, replacing **HEAD** with **GET** to reveal the entire request and message body.

Complete Exercise 8-14 in Student Workbook

Configure Apache and Setup Host-based Access Controls

18 Security Packages and Patches

The Redhat Package Manager (RPM) installs, removes, upgrades, queryies, and verifies Linux security fixes in the form of patches and packages. Most Linux destributions use the RPM.

A package is identified by the <name> <version> <release> and <architecture>. For instance, *xinetd-2.3.15-13.el7.x86_64.rpm,* includes the package name (*xinetd*), version (*2.3.15*), release (*13.el7*), and architecture (*x86_64*). Obtaining this information is a first step in determining the version level of patch applications and further identifying if the patch is up-to-date.

A key element to patch security is ensuring an authorized developer signed a package. When installing or updating a patch or package, signatures are automatically checked. Rpm's use Gnu Privacy Guard (GnuPG) signatures to ensure a downloaded package is trustworthy. GnuPG is a tool for secure communication and installed by default during CentOS installation. To verify a patch signature and ensure a package is not corrupt or tampered with, use **rpm** `with the` **-K** (check signature) and **-v** (verbose) options to print verbose signature information. The first example uses only the **-K** option.

Syntax: **rpm** *[options]* **--nosignature** *<path-to-rpm-file>*

```
[root@CentOS6_A ~]# rpm -K --nosignature /opt/Packages/tree-1.6.0-10.el7.x86_64.rpm
/opt/Packages/tree-1.6.0-10.el7.x86_64.rpm: sha1 md5 OK
```

A message returns listing the algorithm used and 'OK', meaning the file was not corrupt.

Using **-Kv** options also prints a verbose result used for debugging.

```
[root@CentOS6_A ~]# rpm -Kv --nosignature /opt/Packages/tree-1.6.0-10.el7.x86_64.rpm
/opt/Packages/tree-1.6.0-10.el7.x86_64.rpm: sha1 md5 OK
   Header SHA1 digest: OK (a09f99f73ee3f3352489d734c63c32fa41b1be56)
   MD5 digest: OK (c5194b194ecbd871ffbb516de83bfb7d)
```

RPM installs a single downloaded package file. However, it errors out when installing a package that has other dependencies if they are not present. Installing the initial package may require additional package installations.

The Yellowdog Updater Modified (YUM) is a front-end tool for Linux distributions that uses the RPM format. YUM installs a full application and any dependencies needed for it to run (libraries, other applications, etc). If dependencies are not present, YUM searches available repositories locally and across mirror sites on the web. If found, they are presented for consent to install.

To summarize:

- RPM is a package manager and YUM is the front-end used with RPM
- RPM is unable to track dependencies, YUM can and does

Package and patch management using the **rpm** and **yum** commands:

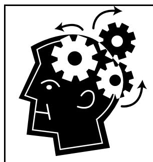
In the examples below, if the command contains an argument the command and option applies to only that argument (package or patch). If it does not contain an argument, the command and option apply to "all" (all packages or patches).

rpm -qa	Query (list) all installed packages
rpm -qi *pkg*	Query installed package information: (Name, version, release, …)
yum list	List installed and available packages in yum repository(s)
yum info	List installed and available package info: (Name, version, release, …)
yum list updates	List only packages with updates available
yum install *pkg [pkg2][…]*	Install package(s)
yum check-update	List all packages requiring updates and obsolete packages
yum update *[pkg1][pkg2][…]*	Updates one or more installed packages
yum update --security	Updates security relevant patches only

Listed below are some best practices of CentOS Patching:

- Develop an inventory of all systems including at a minimum: OS type, OS version, and IP.
- Schedule regular patching of at least once a month.
- Schedule automatic patching, if possible, with a yum-cron job.
- Download patches from trusted sources.
- Prioritize patches based on severity.
- Test patches on standalone systems first, when possible.
- Schedule a maintenance time for installation of patches.
- Make a complete backup of the system before applying patches.

Linux versions like Ubuntu cannot use RPM or YUM and instead use a Debian package manager, Advanced Package Tool (APT), and command, **`apt-get`**. APT contains a set of tools to install, remove, and easily change packages at the command line.



Complete Exercise 8-15 in Student Workbook

Identify OS Version and Patch Level

Objectives

Day 7

At the end of this training day, students will:

- ❖ Develop skills to apply manual file integrity checking on files using hash algorithms.
- ❖ Identify how a HIDS such as Tripwire, monitors a host to help ensure file integrity.
- ❖ Implement IPtables as a Linux firewall, for added security.
- ❖ Identify and avoid network security weaknesses by securing necessary ports.

Exercises

This training day includes the following exercises:

- ❖ Exercise 8-16, Message Digest 5 (MD5) Lab
- ❖ Exercise 8-17, Tripwire
- ❖ Exercise 8-18, IPtables
- ❖ Exercise 8-19, Network Hardening with Nmap

JCAC-eTC

Complete self-study activities and assignments in the online training environment.

19 Host-based Intrusion Detection System (HIDS)

A Host-based Intrusion Detection System (HIDS) monitors system internals for changes made to files, directories, user accounts, and other system changes. Information containing the monitored criteria is included in a database for later review. There are many HIDS provided at cost with support or free through open-source. Some software, such as nmap and security onion can monitor network and host-based intrusion attempts. Linux also provides an Open-Source HIDS SECurity (OSSEC) system that can be tailored to perform analysis, integrity checks, rootkit detection, time-based alerting, and active response. OSSEC is covered in depth in a later topic.

File Integrity Monitoring (FIM) is a HIDS technology used to scan, analyze, and report on unexpected changes to important files. Deviations from what is assumed to be the norm are further diagnosed to determine if the file was tampered with or corruputed. This type of file monitoring verifies and validates files by comparing the latest version of a file to a known, trusted, baseline, providing a critical layer of file, data, and application security. This also aids in the acceleration of incident response and remediation.

20 File Integrity Checking

In a Linux environment, configuration files are normally more exposed as a part of the overall file system, making Linux more vulnerable to direct attacks. Cyber attackers can easily inject malicious code by updating and replacing core files. Some areas that should be audited for change control are the boot loader, kernel parameters, daemons and services, run commands, cron jobs, profiles, and host files. Checking the integrity of associated files helps detect system changes.

A manual file integrity check can be accomplished and verified using the **md5sum** hashing tool. This message digest tool generates a 128-bit hash, creating a digital fingerprint of the file's contents. Later, when **md5sum** is run against the file, the resulting hash is compared with the previously produced hash. If the output is different, the file contents were altered.

```
[root@CentOS6_A ~]# md5sum /usr/bin/gcc
276040c8c82f83e5d5g1312368123122
```

The example above displays hashes for */usr/bin/gcc*. The output can be compared against a known good system or the NSRL (National Software Reference Library) listing of hashes for that package. Other available message digest tools are located in */usr/bin*.

Complete Exercise 8-16 in Student Workbook

Message Digest 5 (MD5) Lab

21 Tripwire

Tripwire is a Linux host-based IDS used to monitor and send alerts on identified filesystem changes. Tripwire scans the file system as instructed by a configuration file, */etc/tripwire/tw.cfg,* and policies set forth in */etc/tripwire/tw.pol*.

A key step in uniquely identifying and creating Tripwire files is initializing the Tripwire database, **`tripwire --init`**. After changes are made to the text versions of the configuration and policy files, the **`twadmin`** command is used to convert the text files into a secure, cryptographically signed versions that should not be edited. The secure versions are used to set-up Tripwire.

21.1 Configuration File

The configuration file stores system-specific information, including the location of Tripwire data files and settings used to send email notifications. These settings are generated during install but can be changed by *root*. The configuration file is signed with a site key and site passphrase, which are required to edit the file. Site key and passphrase are established during Tripwire initialization.

Because Tripwire uses an encrypted configuration file, the *root* user must edit a plain text version of the file and then convert it to an encrypted version for use. During installation, a signed Tripwire configuration file *tw.cfg* is created in */etc/tripwire*, and a plaintext copy of this configuration file, *twcfg.txt,* is generated in the same directory.

21.2 Policy File

The policy file describes system objects to monitor and identifies what properties for each object should be collected and stored in the database file. Each object in the policy file is associated with a property mask that describes what changes to a file should be monitored and which ones can be ignored. During installation, an encoded and signed policy file *tw.pol* is created in */etc/tripwire*, and a plain text copy of this policy file, *twpol.txt,* is generated in the same directory.

Once the policy file is set, the next step is to accomplish an integrity check with **`tripwire --check`**. If errors are generated that should not be reported but need to be added to the database, the database is updated with **`tripwire --update`** to reflect the changes. This prevents the error from being reported during the next check. At the end of an integrity check, a report is produced that can be viewed, saved, and even sent to an off-site mail recipient. after finding a tampered file.

21.2.1 Property Masks

The property mask consists of a series of single-character symbols or select flags, each of which may be preceded with a **+** or **-** sign. If preceded by a **+**, checking is done for that property. If preceded by a **-**, checking is not done for that property. For example, the select flag (*p*) stands for *Permissions and file mode bits*. An object followed by a *+p* has its permissions evaluated to see if they changed. An object followed by a *-p* has its permissions ignored, even if they changed.

Policy rule format is: *object_name* -> *property_mask*. *Object_name* is the full pathname, and *property_mask* specifies what properties of an object to examine or ignore. The "->" separates the path and the property mask. A semi-colon must terminate the rule.

21.2.2 Hash Functions

Tripwire also applies one or more cryptographic hash functions on the file(s). When changes are detected, only the hash value is stored in the database, instead of the entire file contents. Cryptographic hash functions produce a digital signature of the state of the file. Later, the same files are scanned and compared to the original files stored in the initial database. Table 10 lists select flags, available cryptographic hashes, and templates.

Table 10. Tripwire flags, cryptographic hashes, and templates.

Flags	**Controls and Templates**
(a) Access timestamp	(-) Ignore the following properties
(b) Number of blocks allocated	(+) Record and check the following properties
(c) Inode timestamp (create/modify)	Dynamic +pinugtd-srlbamcCMSH
(d) ID of device on which inode resides	Growing +pinugtdl-srbamcCMSH
(g) File owner's group ID	Device +pugsdr-intlbamcCMSH
(i) Inode number	IgnoreAll -pinugtsdrlbamcCMSH
(l) File is increasing in size (a "growing file")	IgnoreNone +pinugtsdrbamcCMSH-l
(m) Modification timestamp	ReadOnly +pinugtsdbmCM-
(n) Number of Links (inode reference count)	
(p) Permissions and file mode bits	(C) CRC-32 Check
(r) ID of device pointed to by inode. (dev objs)	**Cryptographic Hashes**
(s) File size	(H) Haval hash value
(t) File type	(M) MD5 hash value
(u) File owner's User ID	(S) SHA hash value

Properties of the cryptographic hashes included with Tripwire are:

(H)Haval	A cryptographic hash function that produces hashes in different lengths, varying from 128 to 256 bits. Weak cipher.
(M)MD5	Widely used 128-bit hash value, expressed as a 32-digit hexadecimal number, commonly used to check file integrity.
(S)SHA	Secure Hash Algorithm. A set of hash functions (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512) designed by NSA. Popular one-way hash used to create digital signatures. The federal PKI policy recommendation for hashes used with DSA signatures.

Tripwire also supports a 32-bit Cyclic Redundancy Check.

(C)CRC-32	Cyclic Redundancy Check is an error-detecting code that checks chunks of raw data for error and (accidental) changes. Very popular as a built-in check-sum.

Tripwire policy supports variables for string substitution that are defined anywhere between rules. The syntax for a variable definition is: *variable = value*; Variable substitution is anywhere a string could appear. The syntax is: *$(variable).* Tripwire predefines many variables and may not be changed. In the table above, a *ReadOnly* variable has a value of *+pinugtsdbmCM-rlacSH*.

Complete Exercise 8-17 in Student Workbook

Tripwire

22 Security Posture (IPtables)

Most versions of Linux come with a built-in firewall referred to as IPtables. IPtables is a user module accessed via CLI to enter firewall rules into predefined tables. Netfilter is a kernel module that filters, and is considered a first match engine. The firewall is a combination of IPtables and netfilter.

IPtables use IP addresses, protocols, and ports to determine how traffic is filtered. IPtables place rules into a set of predefined chains, INPUT, OUTPUT, and FORWARD, that are compared to network packets relevant to those chains. A determination is then made of what to do with each packet based on the outcome of those rules. These actions are often referred to as targets. The two most common predefined actions are DROP and ACCEPT. Figure 14 illustrates this process.

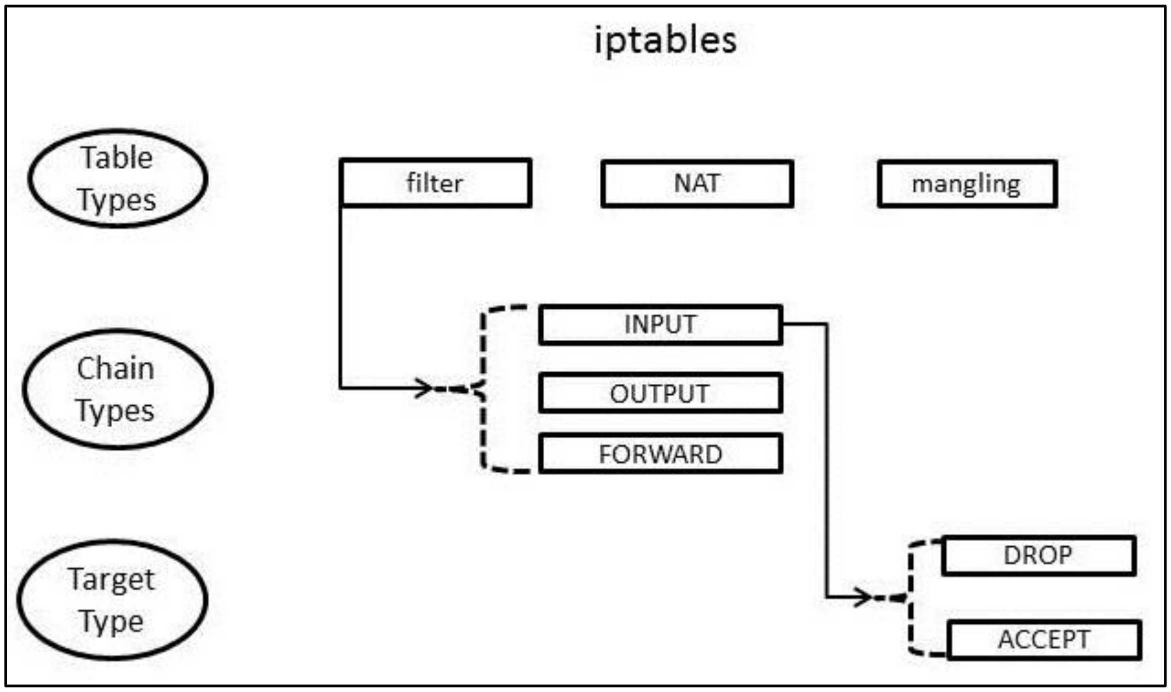


Figure 14. IPtables

22.1 Table Types

Using IPtables, an administrator can set up, maintain, and view the tables of IPv4 packet filter rules in the Linux kernel. As shown below, there are three table types available:

Filter	The filter table is used to filter packets passing through the firewall. This table is the focus. Many tables may be defined and each table may contain a number of built-in chains. Each table may also contain user-defined chains such as:

INPUT	Filter incoming packets
OUTPUT	Filter outgoing packets
FORWARD	Filter packets routed through host computer

NAT	The NAT table is used when changing the source of the IP and contains three types of chains.

PREROUTING	Change IP before forwarding takes place
POSTROUTING	Change IP after forwarding takes place
OUTPUT	Filter on outgoing

Mangling	The mangling table is used to modify packets. For more information on this table, see the man pages.

22.2 Chains and Targets

A chain is a list of rules matched against a set of packets. The rules specify what to do with packets that match. The match is called a target or action. All chains must have a policy of DROP or ACCEPT. This default policy is defined with the **-P** (policy) option of the **iptables** command.

Once a rule is matched and an action is taken, the packet is processed according to the matched rule and no further processing occurs. If the packet gets through all the rules in the chain and reaches the end without matching any rules, the default action for that chain is taken. This action then becomes the default policy and can be set to either ACCEPT or DROP.

For example, the following command defines the default policy for the INPUT chain:

```
[root@CentOS6_A ~]# iptables  -P  INPUT  ACCEPT
```

Rule parameters like **-j** (jump) can be added, allowing the policy to have any of the following values: ACCEPT, DROP, REJECT, DENY, LOG. If a rule matches a packet and its target is ACCEPT or DROP, no further rules are consulted. The REJECT policy is the same as the DROP policy; however, it sends a reply back to the sending host versus the DROP policy, which sends no reply. A DENY policy is the same as the DROP policy.

An INPUT chain identifies filtered packets entering the machine. INPUT chain default policy is normally set to DROP all packets and then rules are added to specifically allow packets from trusted IPs or for certain ports where known services are running (e.g., FTP, Web, or Samba). OUTPUT chains are normally set to ACCEPT all packets with specific DROP rules added as needed since outgoing traffic is normally trusted traffic.

Establishing basic firewall policies provides a foundation for creating more detailed, user-defined rules. When creating the iptables ruleset, order is very important. For example, if one chain is set to drop packets coming from the 192.168.0.0/24 subnet, and another chain is appended to accept all packets from 192.168.0.82, the appended rule is ignored. To resolve this issue, set a rule to ACCEPT 192.168.0.82 first, and then set a catch-all rule to DROP everything else on the subnet.

22.3 Creating Simple Rule Sets

The **`Iptables`** command is used to create the rule sets needed to manage the packets.

Options:	**`-L`**	List rules added in verbose mode.
	`-F`	Flush all existing rules.
	`-P`	Sets the default policy on the specified chain.
	`-A`	Append or add a rule to a specific chain.
	`-j`	Jump to the target of the rule.
	`-i`	Interface.
	`-D`	Delete rule
	`-p`	Protocol

Syntax: **`iptables`** *`[options]`*

List rules in a chain:

```
[root@CentOS6_A ~]# iptables -L
Chain  INPUT  (policy ACCEPT)
target         prot        opt         source            destination

Chain FORWARD (policy ACCEPT)
target         prot        opt         source            destination

Chain OUTPUT (policy ACCEPT)
target         prot        opt         source            destination
```

Flush existing rules so new rules can be added:

```
[root@CentOS6_A ~]# iptables -F
```

Set default policy on INPUT chain:

```
[root@CentOS6_A ~]# iptables -P  INPUT DROP
```

Append rule to INPUT chain:

This chain allows SSH connections over tcp port 22. The protocol (**-p**) allowed is TCP and the destination port (**--dport**) is 22 (SSH).

```
[root@CentOS6_A ~]# iptables -A INPUT -i eth0 -p  tcp --dport 22 -j ACCEPT
```

Delete a single iptable rule:

When multiple lines are given a single line can be deleted based on its number and chain. The following command deletes the 4th chain which is an INPUT chain.

```
[root@CentOS6_A ~]# iptables –D INPUT 4
```

Iptables uses the policies (**-P**) option to create default rules. For instance, the rules listed below blocks all incoming and outgoing packets on a network gateway. Also listed is the rule to block forwarded packets, e.g., network traffic that is routed from the firewall to its destination.

```
[root@CentOS6_A ~]# iptables -P INPUT DROP
[root@CentOS6_A ~]# iptables -P OUTPUT DROP
[root@CentOS6_A ~]# iptables -P FORWARD DROP
```

This is a good start, but in order to allow users to perform network-related tasks and use network related applications, certain ports must be opened for communication. For example, opening up port 80 to allow Internet access is needed. The examples below open port 80 on the firewall for INPUT and OUTPUT.

```
[root@CentOS6_A ~]# iptables -A INPUT -i eth0 -p tcp --dport 80 -j ACCEPT
[root@CentOS6_A ~]# iptables -A OUTPUT -p tcp --sport 80 -j ACCEPT
```

A breakdown of the command, options, and switches needed to allow access to port 80 on the firewall is shown here:

-A INPUT	Append the rule to chain (INPUT)
-p tcp	Protocol TCP (implied)
-m tcp	Protocol TCP (matched)
--sport 80	Source port 80
--dport 80	Destination port 80
-j ACCEPT	Jump to target (specifies the target of the rule)

22.4 Scripting IPtables

Changes made to firewall rules are only valid for the time the computer is running. If the system is rebooted, rules are automatically flushed and reset to their original policy state. To save the rules so they are loaded later, use the following command:

```
[root@CentOS6_A ~]# /sbin/service iptables save
```

The rules are stored in the file */etc/sysconfig/iptables* and are applied when the service is started or restarted, including when the system is rebooted.

Typing all the commands at the command line every time a change is needed or whenever the system is restarted can become tedious. The best way to deal with IPtables is by creating a script that can run every time the system starts up, or for that matter, any time desired. Below is an example of an IPtable script. Create a file called *myfirewall.sh* and enter the following information exactly as shown:

```
#!/bin/bash
#
#  This is an example of an IPtable configuration script
#  Flush out all rules and start new
iptables  -F
#  Set the default policies
iptables  -P  INPUT   ACCEPT
iptables  -P  FORWARD ACCEPT
iptables  -P  OUTPUT  ACCEPT
#  Set access for the localhost
iptables  -A  INPUT  -i  lo  -j  ACCEPT
#  Allow SSH connection on tcp port 22
iptables  -A  INPUT  -p  tcp  --dport  22  -j  ACCEPT
#  Save the settings
/sbin/service  iptables  save
#
#  List the rules to see what was done
iptables  -L  -v
```

Make the script executable:

```
[root@CentOS6_A ~]# chmod 770 myfirewall
```

Now the script can be edited and run from the command line at any time.

```
[root@CentOS6_A ~]# ./myfirewall
```

Complete Exercise 8-18 in Student Workbook

IPtables

23 Network-based Security using Nmap

Network-based security addresses potential vulnerabilities on a network. Nmap is a network discovery and port scanning tool used for security auditing on local and remote networks. Nmap provides the following features depending on options used:

- Host discovery: Identify potential hosts that respond to network requests, e.g. ping sweep
- Port scanning: Scan and discover ports on specific networks and hosts
- OS detection: Identify OS name, version, and network details
- Application version detection via banner grabbing: What apps are running and their version, i.e., Telnet or FTP

Nmap uses raw IP packets to determine what hosts are available on the network, what services those hosts are offering, what OSs they are running, types of packet filters/firewall are in use, and other characteristics.

Nmap sends ICMP and ARP packets to all addresses in a range to reveal available hosts. Hosts are scanned based on a specific port, individual port(s), or a port range. Additional options provide the OS version of hosts and may also perform a port scan to detect the version of service running on those ports. Based on the results, a system administrator can track down services using those ports and properly secure them.

Syntax: **nmap** *[options] <ip address>*

Options:		
	-sn	Ping Scan (Available hosts but no port scan)
	-sS	SYN TCP scan (Stealthy, requires *root* user)
	-sT	TCP connect scan
	-p *<port>*	Specify port(s)
	-sV	Service version detection
	-O	Enable OS Version detection (make a guess)
	-O --osscan-guess	Guess more aggressively
	-D	Decoy scan (perform a scan with spoofed IP(s))
	-D RND:10	Randomly generate 10 decoy IP addresses

Example 1: Ping Sweep an IP address range, available hosts are displayed.

```
[root@CentOS6_A ~]# nmap -sn 192.168.0.0-255
Starting Nmap 5.51 (http://nmap.org) at 2019-02-01 12:34 CST
Nmap scan report for 192.168.0.51
Host is up (0.00032s latency).
MAC Address:  00:0C:29:9E:FC:13 (VMware)
Nmap scan report for 192.168.0.70
Host is up (0.00011s latency).
MAC Address:  00:0C:29:1F:43:7D (VMware)
Nmap scan report for CentOS6_A.jcac.local (192.168.0.80)
```

Example 2: Host discovery and open ports for a specific IP.

```
[root@CentOS6_A ~]# nmap -sS 192.168.0.80
Starting Nmap 5.51 (http://nmap.org) at 2019-02-01 12:34 CST
Nmap scan report for CentOS6_A.jcac.local (192.168.0.80)
Host is up (0.00000010s latency).
Not shown: 995 closed ports
PORT        STATE  SERVICE
21/tcp      open   ftp
22/tcp      open   ssh
23/tcp      open   telnet
25/tcp      open   smtp
111/tcp     open   rpcbind
Nmap done: 1 IP address (1 host up) scanned in 0.04 seconds
[root@CentOS6_A ~]#
```

Example 3: Host Discovery and open ports for a sub-net.

```
[root@CentOS6_A ~]# nmap -sT 192.168.0.0/24
Starting Nmap 5.51 (http://nmap.org) at 2019-02-01 12:34 CST
Nmap scan report for 192.168.0.51
Host is up (0.00000010s latency).
Not shown: 997 closed ports
PORT        STATE  SERVICE
135/tcp     open   msrpc
139/tcp     open   netbios-ssn
445/tcp     open   Microsoft-ds
MAC Address: 00:0C:29:9E:FC:13 (VMWARE)

Nmap scan report for CentOS6_B.jcac.local (192.168.0.81)
Host is up (0.00000010s latency).
Not shown: 999 closed ports
PORT        STATE  SERVICE
22/tcp      open   ssh
MAC Address: 00:0C:29:39:6D:E3 (VMWARE)
Nmap done: 1 IP address (1 host up) scanned in 5.18 seconds
<ouput truncated>
[root@CentOS6_A ~]#
```

Example 4: Decoy scan. Target receiving the request sees mulitlple spoofed IP addresses.

```
[root@CentOS6_A ~]# nmap -D RND:10 192.168.0.80
Host is up (0.000081s latenc).
Not shown: 996 closed ports
PORT        STATE       SERVICE
22/tcp      open        ssh
23/tcp      open        telnet
MAC Address: 00:0C:29:1F:43:7D (VMware)
<Output truncated>
```

Example 5: Obtain service version.

```
[root@CentOS6_A ~]# nmap -sV 192.168.0.81
Starting Nmap 5.51 (http://nmap.org) at 2019-02-01 12:34 CST
Nmap scan report for CentOS6_B.jcac.local (192.168.0.81)
Host is up (0.00000010s latency).
Not shown: 999 filtered ports
PORT        STATE SERVICE     VERSION
22/tcp      open  ssh         OpenSSH 5.3 (protocol 2.0)
MAC Address: 00:0C:29:39:6D:E3 (VMWARE)
Service detection performed.
Nmap done: 1 IP address (1 host up) scanned in 5.18 seconds
[root@CentOS6_A ~]#
```

Example 6: Aggresively guess OS version.

```
[root@CentOS6_A ~]# nmap -O --osscan-guess 192.168.0.81
Starting Nmap 5.51 (http://nmap.org) at 2019-02-01 12:34 CST
Nmap scan report for CentOS6_B.jcac.local (192.168.0.81)
Host is up (0.00000010s latency).
Not shown: 996 filtered ports
PORT        STATE SERVICE     VERSION
22/tcp      open  ssh         OpenSSH 5.3 (protocol 2.0)
MAC Address: 00:0C:29:39:6D:E3 (VMWARE)
Device type: WAP|generao purpose|<ouput truncated>
Running (JUST GUESSING): Netgear embedded (96%), Linux
2.6.X|2.4.X (92%), Crestron 2-Series (91%), Check Point embedded
(91%), AXIS Linux 2.6.X (91%), Citrix Linux 2.6.X (89%), IBM
embedded (88%), Linksys embedded (88%)
Aggressive OS guesses: Netgear DG834G WAP (96%), Linux 2.6.24 -
2.6.35 (92%), Linux 2.6.32 (92%), Linux 2.6.9 - 2.6.18 (92%),
Crestron XPanel control system (91%), Linux 2.4.26 (Slackware
10.0.0) (91%), Check Point VPN-1 UTM appliance (91%), AXIS 211A
Network Camera (Linux 2.6) (91%), AXIS 211A Network Camera (Linux
2.6.20) (91%), Linux 2.6.24 (90%)
<output truncated>
OS detection performed.
Nmap done: 1 IP address (1 host up) scanned in 5.18 seconds
[root@CentOS6_A ~]#
```

Complete Exercise 8-19 in Student Workbook

Network Hardening with Nmap

Objectives

Day 8

At the end of this training day, students will:

- ❖ Capture and analyze network traffic using tcpdump.
- ❖ Secure network service banners to reduce enumeration.
- ❖ Implement TCP daemons to add additional security to selected services.
- ❖ Demonstrate the ability to restrict root access on a Linux system.
- ❖ Comprehend the rsyslog facility and its configuration files.
- ❖ Identify Linux key log files.

Exercises

This training day includes the following exercises:

- ❖ Exercise 8-20, Packet Sniffing using TCPDump
- ❖ Exercise 8-21, Banner Security
- ❖ Exercise 8-22, TCP Wrappers
- ❖ Exercise 8-23, Console Logins
- ❖ Exercise 8-24, Log Files and Auditing

JCAC-eTC

Complete self-study activities and assignments in the online training environment.

24 Identifying Hardware and Software

Packet sniffing enables administrators to detect and monitor devices located on their network based on specific traffic patterns. Packet sniffing software shows devices using normal services on their respective port numbers and can indicate anomalous network/protocol activity. Linux provides a packet-sniffing tool as a part of its default install called TCPdump. TCPdump is a common CLI tool used to capture network packets.

`tcpdump` Prints a description of network packet contents, matching a Boolean expression.

Options and syntax:

`-D` Print a list of network interfaces available.

```
[root@CentOS6_A ~]# tcpdump -D
```

`-i` *<interface>* Listen on interface; See all activity on device.

```
[root@CentOS6_A ~]# tcpdump -i eth4
```

`-A` Print each packet in ASCII format

```
[root@CentOS6_A ~]# tcpdump -i eth4 -A
```

`-e` Print link-level header on each line.

```
[root@CentOS6_A ~]# tcpdump -i eth4 -e
```

`-X` Print all packet in ASCII and Hex format no link-level header

```
[root@CentOS6_A ~]# tcpdump -i eth4 -X
```

`-XX` Print packet headers data, and link-level header (Ethernet)

```
[root@CentOS6_A ~]# tcpdump -i eth4 -XX
```

`-c` *<count>* Print the first 30 packet in ASCII and Hex format then exit.

```
[root@CentOS6_A ~]# tcpdump -i eth4 -c 30 –XX
```

`-n` Do not resolve hostnames (**-nn** Do not resolve hostnames or port names)

```
[root@CentOS6_A ~]# tcpdump -i eth4 -c 30 -n -XX
```

`-w` *<file>* Write raw packets to a file vice parsing and printing them.

```
[root@CentOS6_A ~]# tcpdump -i eth4 -c 20 –w
 /tmp/mycap.pcap
```

`-r` *<file>* Read packets from a file.

```
[root@CentOS6_A ~]# tcpdump -r /tmp/mycap.pcap -XX
```

`-v` Verbose

```
[root@CentOS6_A ~]# tcpdump -v -r /tmp/mycap.pcap
```

[expressions]	Berkely Packet Filtering (BPF) selects packets to dump. If no expression is given, dump all packets on the net. Otherwise, dump only packets where the expression is true. See Table 11 below.

Table 11. BPF expressions.

`type`	host, net, or port
`direction`	src, dst, src or dst, src and dst
`protocol`	ether, ip, tcp, udp, arp, rarp
`logical operator`	and, or, not
`dst host` *<host>*	destination field of packet is *<host>*
`src host` *<host>*	source field of packet is *<host>*
`host` *<host>* **`either`**	Source/ destination of packet is *<host>*
`ether dst` *<ehost>*	ethernet destination address is *<ehost>*
`ether src` *<ehost>*	ethernet source address is *<ehost>*
`ether host` *<ehost>*	either ethernet source/ destination address is *<ehost>*
`dst port` *<port>*	packet is ip/tcp, ip/udp with a destination port value of *<port>*
`src port` *<port>*	packet has a source port value of *<port>*
`port` *<port>*	either source/ destination port of packet is *<port>*
`tcp src port` *<port>*	matches only tcp packets whose source port is *<port>*

When writing raw packets to a file, all packets identified by the BPF expressions are captured. The naming convention often ends with *.pcap*, making it easy for a protocol analyzers such as Wireshark to translate. When reading packets from a file, options are used to extract only those packets desired, format required, and the level of verbosity. BPF expressions should normally follow any command options. Command options must coincide with their value, i.e., -c 50.

Commonly used options and filters:

Listen on eth4 for traffic going to or from host apache. Print results in ASCII.

```
[root@CentOS6_A ~]# tcpdump  -i  eth4  -A  host  apache
```

Listen on eth4, capture first 40 packets to or from port 80.

```
[root@CentOS6_A ~]# tcpdump  -i  eth4  -c 40 port  80
```

Listen on eth4 for traffic with a destination to 192.168.0.82. Print results in Hex and ASCII.

```
[root@CentOS6_A ~]# tcpdump  -i  eth4  -XX  dst host  Apache
```

Capture packets on eth4. Destination host is 192.168.0.81 AND source/destination port is 23.

```
[root@CentOS6_A ~]# tcpdump  -i eth4 dst host 192.168.0.81 and port 23
```

Read packets captured with max details from a file named */tmp/mycap.pcap*.

```
[root@CentOS6_A ~]# tcpdump -vvv -r /tmp/mycap.pcap
```

Listen on interface eth4. Capture all data packets coming across the sendmail port and write them to /tmp/mycap2.pcap.

```
[root@CentOS6_A ~]# tcpdump  -i  eth4  -w /tmp/mycap2.pcap port  25
```

Record a capture of 20 packets to a file named */tmp/mycap.pcap*.

```
[root@CentOS6_A ~]# tcpdump -i eth4 -c  20  -w /tmp/mycap.pcap
```

Listen on interface eth4. Capture any packets where tcp src port is ssh.

```
[root@CentOS6_A ~]# tcpdump -i  eth4  tcp  src  port  22
```

Capture all traffic on eth4 that is not ICMP.

```
[root@CentOS6_A ~] # tcpdump -i eth4 dst 192.168.0.80 and not icmp
```

Capture all traffic on eth4 from source 192.168.0.80, but not Telnet traffic, in verbose mode.

```
[root@CentOS6_A ~] # tcpdump -i eth4 -v src 192.168.0.80 and not dst port 23
```

Acquiring and translating raw packet data from IP and TCP headers can be used to map a network. Identifying key data fields in a header was introduced in the Networking Concepts and Protocols module. Information sheets used in that module are supplied at the end of this module to enforce those concepts and use as a quick reference. These header fields remain relevant.

Using the **–xx** option is vital when determining source/target IPs, ports, and macs.

See Information Sheet 8-2 in Student Workbook

IPv4 Header and TCP Header Key Fields

Complete Exercise 8-20 in Student Workbook

Packet Sniffing using TCPDump

24.1 Network Service Banners

The first step in the penetration testing process is planning and reconnaissance. Many network services use banners that reveal specific OS information. Hackers use a reconnaissance technique known as "banner grabbing" to launch known exploits against a system. The *telnet, ssh,* and *sendmail* banners are discussed below.

24.1.1 Telnet Banner

Using Telnet to access a system may display OS banner information. Even an unsuccessful Telnet attempt can reveal OS version. A successful login may reveal detailed OS information. The two bolded lines below identify the default Telnet banner and are displayed prior to login.

```
[root@CentOS6_A ~]#  telnet 192.168.0.80
Trying 192.168.0.80 . . .
Connected to 192.168.0.80
Escape character is '^]'.
CentOS release 6.3 (Final)
Kernel 2.6.32-279.el6.x86_64 on an x86_64
Login: student
Passwd: ********
Last login: Tue Oct 29 13:46:06 from localhost
*** This system is being monitored ***
[student@CentOS6_A ~]$
```

Banners are secured by hiding or modifying the bolded information displayed in the output of the telnet command above. The italicized line is the message of the day (motd) located in */etc/motd*, given after a successful login.

The default Telnet banner in CentOS is identified in */etc/issue*. Within the file are two lines of information presented to the user during login. The first line displays the OS and Release version, e.g., *CentOS release 6.3 Final*. The second line presents the kernel revision and machine.

/etc/issue should be modified and renamed to */etc/issue.net* to display a preferred networked banner.

24.1.2 SSH Banner

When using **ssh** to securely connect to another system, OS information may be displayed. **ssh** sessions use the same banner information entered in */etc/issue.net* used by **telnet**. To have a specific motd banner appear after an **ssh** login, modify */etc/ssh/sshd_config* and enter the path to the motd entered. By default, the *#Banner none* line is commented out. Remove the *#* from the beginning of this line and replace *none* with */etc/issue.net*. Restart the *sshd* service and the same information provided for **telnet** is provided when logging in with **ssh**.

24.1.3 Sendmail Banner

Sendmail is the Mail Transfer Agent (MTA) used on Linux systems. MTA is an application that routes and transmits electronic mail using the SMTP protocol. Sendmail uses weak security mechanisms but is still commonly used in most Linux variants. Sendmail version information is displayed when connecting to port 25 of a networked host using **telnet**.

```
[root@CentOS6_A ~]#  telnet 192.168.0.83 25
Trying 192.168.0.83 . . .
Connected to 192.168.0.83.
Escape character is '^]'.
220 CentOS7_B.jcac.local ESMTP Sendmail 8.14.4/8.14.4; Thu, 5 Jan 2019 13:09
```

If a "No route to host" error appears, sendmail is not set to accept email over a network. Only local connections over port 25 are allowed. Allow sendmail to accept email over a network by editing */etc/mail/sendmail.cf*. Comment DaemonPortOptions=Port=smtp,Addr=127.0.0.1, Name=MTA". Write and save the configuration file. A restart to sendmail is required for changes to take effect. Run the following command to restart the sendmail service:

```
[root@CentOS7_A ~]# systemctl enable sendmail
```

Exit sendmail with the **<CTRL>]** keys and then type *quit* to leave the Telnet session.

The sendmail banner, Sendmail 8.14.7/8.14.4, can be hidden or modified. This configurable information is located in the sendmail configuration file, */etc/mail/sendmail.cf*. Change the Smtp Greeting message as shown below:

From: `O SmtpGreetingMessage=$j Sendmail $v/$z; $b`

To: `O SmtpGreetingMessage=Mail Server Ready`

Securing Special SMTP Commands

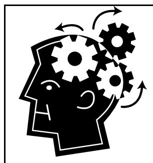
SMTP contains additional commands to verify (vrfy) mail users or expand (expn) a mailing list. Hackers use these commands to verify accounts on a system. Once account names are obtained, password guessing begins. Below, **telnet** is used to access port 25 of 192.168.0.83. A mail user can be identified using the **vrfy** command followed by a random name.

```
[root@CentOS6_A ~]#  telnet 192.168.0.83 25
Trying 192.168.0.83...
Connected to 192.168.0.83.
Escape character is '^]'.
220 CentOS7_A.jcac.local ESMTP Sendmail 8.14.4/8.14.4; <Date/Time> -0600
VRFY root
250 2.1.5 root <root@CentOS7_A.jcac.local>
VRFY student
250 2.1.5 student <student@CentOS7_A.jcac.local>
VRFY sam
550 sam... User unknown
```

Modify the Privacy Flags section of *sendmail.cf* to set the PrivacyOptions. The **VRFY/EXPN** commands can be secured by editing */etc/mail/sendmail.cf*. Append *novrfy* to the end of PrivacyOptions as shown below. Restart sendmail after making changes.

```
   O PrivacyOptions=authwarnings,noexpn,restrictqrun,novrfy
```

With *noexpn* and *novrfy* set, obtaining user name or mailbox expansion results in an error.



Complete Exercise 8-21 in Student Workbook

Banner Security

24.2 TCP Wrappers

TCP Wrappers is a host-based ACL system providing a high degree of control over incoming TCP connections. It allows hostname or IP address, names, or identification query replies to be used as tokens on which to filter for access control. It includes a library called *libwrap*, that implements the functionality by assisting with network filtering.

Originally, only services listed in SystemV based systems using *inetd* could be wrapped with the *tcpd* program. Today, any network service daemon can be linked against libwrap directly, if the service includes the ability to link to libwrap. To determine if *tcpd* supports a service, use **ldd** to print shared library dependencies for that service. For example,

```
[root@CentOS6_A ~]#  ldd /usr/sbin/sshd | grep wrap
```

Access control is configured in */etc/hosts.allow* and */etc/hosts.deny*. *Hosts.allow* is always read first and both files are read from top to bottom. The allow and deny keywords make it possible to keep all access control rules within a single file, i.e., */etc/hosts.allow*. Each line contains up to three fields, with the third field being optional. When only one access control file is used, the optional shell command can be an **ALLOW** or a **DENY** command. The format is as follows:

daemon:	client:	(optional shell command)
1	2	3

1. daemon	Command name of TCP daemons (*sshd, telnetd, vsftpd*, …)
2. client	Specifies hostname or IP address of incoming connection
3. shell_command	Specifies a command to execute if daemon_list and client_host_list match (ALLOW or DENY).

Example of */etc/hosts.allow (optional)*:

sshd: 175.45.101.2	IP 175.45.101.2 is allowed to connect via ssh
vsftpd: sun.net, 175.45.	*sun.net* and 175.45 is allowed to transfer files via vsftp.

Example of */etc/hosts.deny*

All:All	This gives a default deny to any connections not matched in */etc/hosts.allow*.

Using only */etc/hosts.allow* with optional shell commands. (**Preferred Method**)

sshd: 192.168.0.81: ALLOW	Allow 192.168.0.81 ssh access
sshd: ALL: DENY	Deny all others ssh access

Conversely:

sshd: 192.168.0.81: DENY	Deny 192.168.0.81 ssh access
sshd: ALL: ALLOW	Allow all others ssh access

The following steps are taken for setting Access Control using TCP Wrappers.

Step 1. Determine if the *tcp_wrappers* program is installed.

```
root@CentOS6_A ~]# yum list tcp_wrappers
```

Step 2. Install TCP Wrappers if not installed. If installed skip to step 3.

```
root@CentOS6_A ~]# yum –y install tcp_wrappers
```

Step 3. Determine if the daemon/service to be used with *tcpd* exists under TCP Wrapper control. Use the following command to check for a link to the ssh service.

```
[root@CentOS6_A ~]# ldd /usr/sbin/sshd | grep wrap
libwrap.so.0 => /lib64/libwrap.so.0 (0x00007f01b4e2a000)
```

This ssh service is under TCP Wrapper control because it includes "libwrap".

Step 4. Configure */etc/hosts.allow*. The example below denies access to *vsftpd* from 192.168.0.0/24, but allows 192.168.0.81.

```
vsftpd: 192.168.0.0/24: DENY
vsftpd: 192.168.0.81: ALLOW
```

The next two examples use both a *hosts.allow* and a *hosts.deny* file.

This example denies access to *vsftpd* from everyone except 192.168.0.0/24.

```
[root@CentOS6_A ~]# vi /etc/hosts.deny
vsftpd: ALL
[root@CentOS6_A ~]# vi /etc/hosts.allow
vsftpd: 192.168.0.0/24
```

This example denies access to ALL services under TCP Wrapper control except for the jcac.domain and 192.168.0.80.

```
[root@CentOS6_A ~]# vi /etc/hosts.deny
ALL: ALL
[root@CentOS6_A ~]# vi /etc/hosts.allow
ALL: jcac.domain 192.168.0.80
```

Complete Exercise 8-22 in Student Workbook

TCP Wrappers

25 Logs and Auditing

Logs are used to capture events at different levels of importance for activities that occur on a system. These logs include OS, application, device, authentication, and even basic informational logs. Log auditing is used to capture events when something behaved unexpectedly. Linux provides a daemon that constantly monitors the system for generated error or informational messages, and then acts on those messages with notifications. This section introduces *roots* ability to login to a system locally and remotely, and ways of auditing and logging those attempts. Restricting *root* access to a system is introduced and requires a user to **sudo** into a system, to gain privileged access.

25.1 Console Logins

When a user logs into a Linux system, the standard input, stdin, comes from the keyboard and the standard output, stdout and stderr, is redirected to the screen. This is termed a console login. The **tty** command displays the file name of the terminal connected to standard input.

Linux can redirect standard input to get data from another source other than the keyboard and redirect output to display data to a destination other than the screen. A pseudo terminal device (pty) is emulated by other programs i.e., **ssh** and **telnet**. A pts is the slave part of a pty that provides an interface that behaves exactly like a classical terminal. For every terminal (tty) opened, a pts file is created with a 1 up number beginning with 0.

Given a new CentOS6_A session, open a terminal window and execute **tty**. Open a second terminal and exectute **tty** in that terminal. The results are files named */dev/pts/0* and */dev/pts/1*.

/etc/securetty restricts the ability of users to login as *root* on any terminal other than the console. There are two exceptions to this rule: *securetty* must be enabled, and the OpenSSH Suite of tools must not be active. By default, *securetty* is enabled and OpenSSH is used. A blank */etc/securetty* does NOT necessarily prevent *root* user from logging in remotely using the OpenSSH suite of tools, because authentication is provided prior to opening the console. By default, CentOS does not allow *root* access to a Telnet session. To allow access, pts's are added to the end of the */etc/securetty* file. Below is an abbreviated list of entries with pts entries:

```
 [root@CentOS6_A ~]# more /etc/securetty
console
vc/1
vc/2
tty1
tty2
pts/0
pts/1
pts/3
```

Disabling the *root* account does not affect the following programs: `su, sudo, ssh, scp`, or `sftp`. These are a part of the OpenSSH suite of tools. To prevent root access for `ssh, scp`, and `sftp`, modify */etc/ssh/sshd_config* and set *PermitRootLogin* parameter to *no*.

Complete Exercise 8-23 in Student Workbook

Console Logins

25.2 rsyslog

The **rsyslog** facility sends messages generated by the kernel and system utilities to the rsyslogd daemon. Processes are programmed to generate messages at various levels of importance in response to actions taken or conditions encountered. Levels range from simple debugging (debug) to emergency (emerg) levels. This information is invaluable when determining system health and when reviewed regularly, can provide warnings of some types of attacks.

System administrators control the manner that *rsyslogd* manages messages by modifying */etc/rsyslog.conf*. The *rsyslogd* sorts messages by their source or importance and enables administrators to decide what messages are kept and where they are placed. Depending on configuration of */etc/rsyslog.conf*, the rsyslogd daemon can:

- Write messages to a system log (/var/log/messages, /var/log/secure, /var/log/maillog)
- Write messages to the system console or pseudo terminal (/dev/pts/3)
- Forward messages to a list of users (root or student)
- Forward messages to rsyslogd on other hosts over the network (@CentOS6_B)

Each entry in */etc/rsyslog.conf* consists of two tab-separated fields: selector and action.

The **selector** field consists of a facility and a level written as *facility.level*. Facilities represent categories of system processes that can generate messages. Levels represent severity or importance of message. Table 12 below shows a few of the rsyslog selector fields. Table 13 displays the levels of importance for the selectory fields.

Table 12. rsyslog selector field facilities.

kern	Messages generated by kernel.
user	Messages generated by user processes. Default priority for messages from programs or facilities not listed in this file.
mail	Mail system.
daemon	System daemons, such as in.ftpd and in.telnetd.
authpriv	Authorization system including `login`, `su`, and `getty`.
syslog	Messages generated internally by rsyslogd.

lpr	Line printer spooling system – lpr and lpc.
news	Files reserved for USENET network news system.
uucp	Linux-to-Linux copy (UUCP) system; does not use syslog.
cron	Cron and at facilities, including `crontab`, `at`, and `cron`.
local0-7	Field reserved for local use.
mark	Time-stamp messages produced internally by syslogd.
*	All facilities, except for mark facility.

Table 13. Levels of importance for selector field facilities.

Emerg	Panic conditions normally broadcasted to all users.
Alert	Conditions that should be corrected immediately, such as a corrupted system database.
Crit	Warnings about critical conditions, such as hard device errors.
Err	Other errors.
Warn	Warning messages.
Notice	For conditions that are not error conditions, but might require special handling.
Info	Informational messages.
Debug	Messages normally used only when debugging a program.

The action field defines where messages are forwarded and has one of the following forms:

/filename	Absolute path for log file is required.
@host	Messages are forwarded to syslogd of the remote system.
user1, user2	*user1* and *user2* receive messages if they are logged in.
*	All logged-in users receive messages.

Below is an example entry from the */etc/rsyslog.conf*:

```
*.err    /var/log/messages
```

```
Selector Field          *.err
                           Facility (* is a wild card meaning any or all)    *
                           Delimiter                                         .
                           Message Level                                     err

Action Field            /var/log/messages
```

The r*syslog.conf* file should log: debug mail messages to mail_log, error messages to errorlog, informational messages to syslog, and broadcast critical messages to all users logged in.

25.3 Key Log Files

Table 14 defines the key files where Linux stores its logging information. The bold-faced items **last, who,** and **lastlog** are binaries used to view their ASCII component.

Table 14. Linux log information files and commands.

*/usr/bin/**last***	Display login/logout information about users and terminals. Information obtained from accounting information in */var/log/wtmp(x)*.
*/usr/bin/**who***	Lists user's name, tty (pts/5), login time, and related information for each current system user. Information obtained from */var/run/utmp(x)* that contains current user and account info.
*/usr/bin/**lastlog***	Reports the most recent login of all users or a given user. The **`lastlog`** command formats and prints contents of */var/log/lastlog*.
/var/log/secure	Display a record of attempts to execute the **`su`** command.
/var/log/messages	Lists various system warning and error message events as determined by the rsyslogd.
/var/log/maillog	Maintains a record of all **mail** activity
/var/log/cron	Maintains a record of all **`cron`** activity

Complete Exercise 8-24 in Student Workbook

Log Files and Auditing

Objectives

Day 9

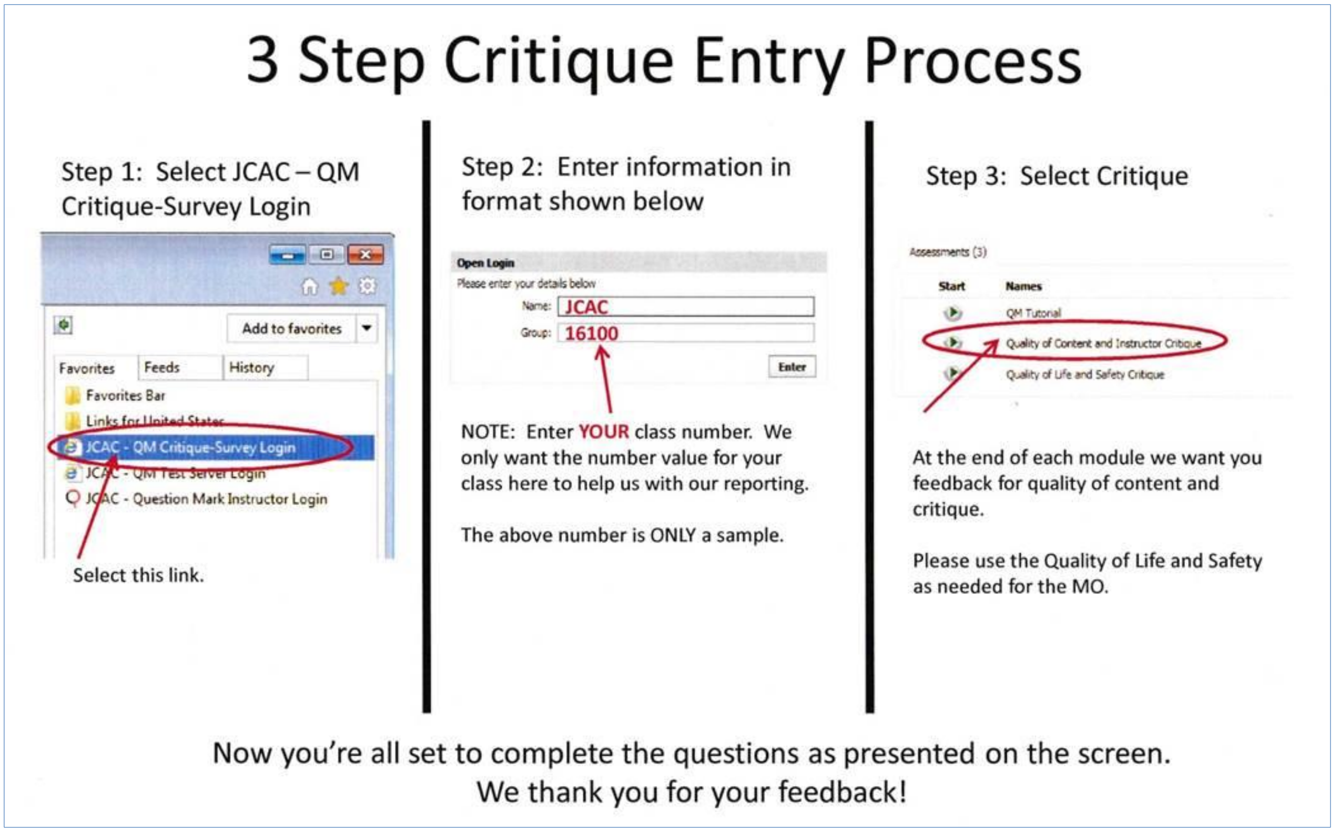
- ❖ None

Exercises

- ❖ None

Critiques

- ❖ Submit JCAC-eTC Critique
- ❖ Submit QM Instructor and Course Critique



3 Step Critique Entry Process

Step 1: Select JCAC – QM Critique-Survey Login

Add to favorites
Favorites
Feeds
History
Favorites Bar
Links for United States
JCAC - QM Critique-Survey Login
JCAC - QM Test Server Login
JCAC - Question Mark Instructor Login

Select this link.

Step 2: Enter information in format shown below

Open Login

Please enter your details below

Name: JCAC

Group: 16100

NOTE: Enter **YOUR** class number. We only want the number value for your class here to help us with our reporting.

The above number is ONLY a sample.

Step 3: Select Critique

Assessments (3)
Start
Names
QM Tutorial
Quality of Content and Instructor Critique
Quality of Life and Safety Critique

At the end of each module we want you feedback for quality of content and critique.

Please use the Quality of Life and Safety as needed for the MO.

Now you're all set to complete the questions as presented on the screen.
We thank you for your feedback!