The Permissions Spectrum

Offensive Computer Security
(FSU Spring 2014)

Hi my name is: root

Outline

- Basics of a system (Linux)
  - BIOS, bootloader, OS
- Rings Model
  - Permissions
- VR
- Linux

The BIOS

Is the ultimate authority of what hardware is and is not installed on a system.

(typically)

:)

Boot Process

BIOS

POST

- CMOS
- Hardware initialization (CPU, Video Card, ...)

Bootloader

OS

Linux boot up

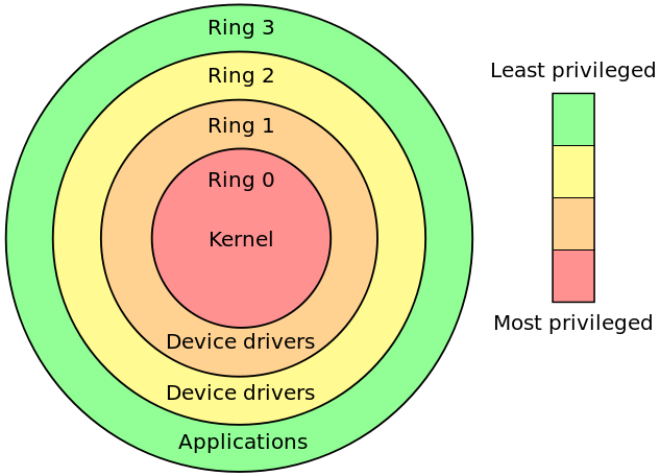
- Boot loader (i.e. GRUB, LILO), then presents the boot options
  - calls start_kernel() on the selected option
  - start_kernel() performs most of the system setup
    - interrupt handling, memory manager, device initialization, drivers
- start_kernel() spawns the idle process, process scheduler, and the **init process** (which is in userspace)

The Kernel

- Handles all operating system processes
  - memory management,
  - task scheduling (context switching)
  - I/O (and CPU interrupts, per packet, keystroke, etc...)
  - interprocess communication
  - and overall system control
- Kernel is typically loaded as an image file, compressed into either *zImage or bzImage* formats (using zlib).
- Linux has a monolithic kernel

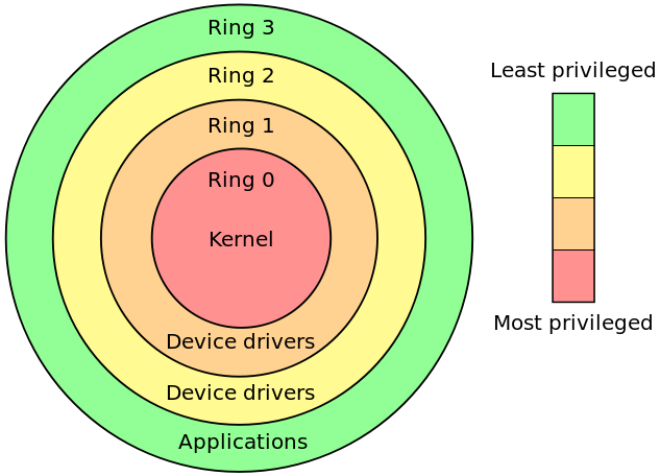
The “Academic” Rings model

- For fault tolerance, and security
- Provide different levels of access
  - 3 Normal non-root user applications
  - 2 Device drivers (keyboard / mice / …)
  - 1 Device drivers (video card, etc)
  - 0 Kernel



The Rings security Model

- Things operate higher than ring 0
  - SMM (-2) (System Management Mode) on intel chips
  - IPMI (-3)
  - BIOS (-1)
- The original Multics had 8 rings!
  - had special register for ring #



The “Practical” Rings model

- For fault tolerance, and security
- Provide different levels of access
  - 4 Sandboxed non-root user applications
  - 3 Normal non-root user applications
  - 0 Kernel / root user
  - -1 BIOS
  - -2 SMM
  - -3 IPMI (Servers)
  - Physical Access

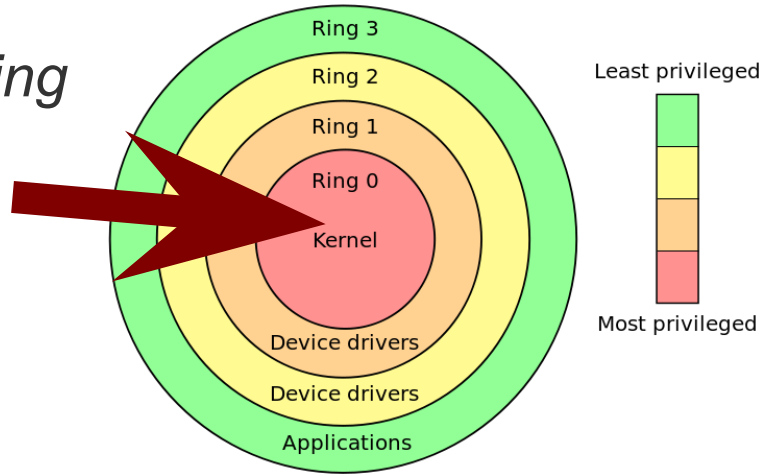
Vulnerability Research

Privilege Escalation:

- *Type of attack resulting in higher (or more) permissions for user/attacker.*

Vulnerability Research:

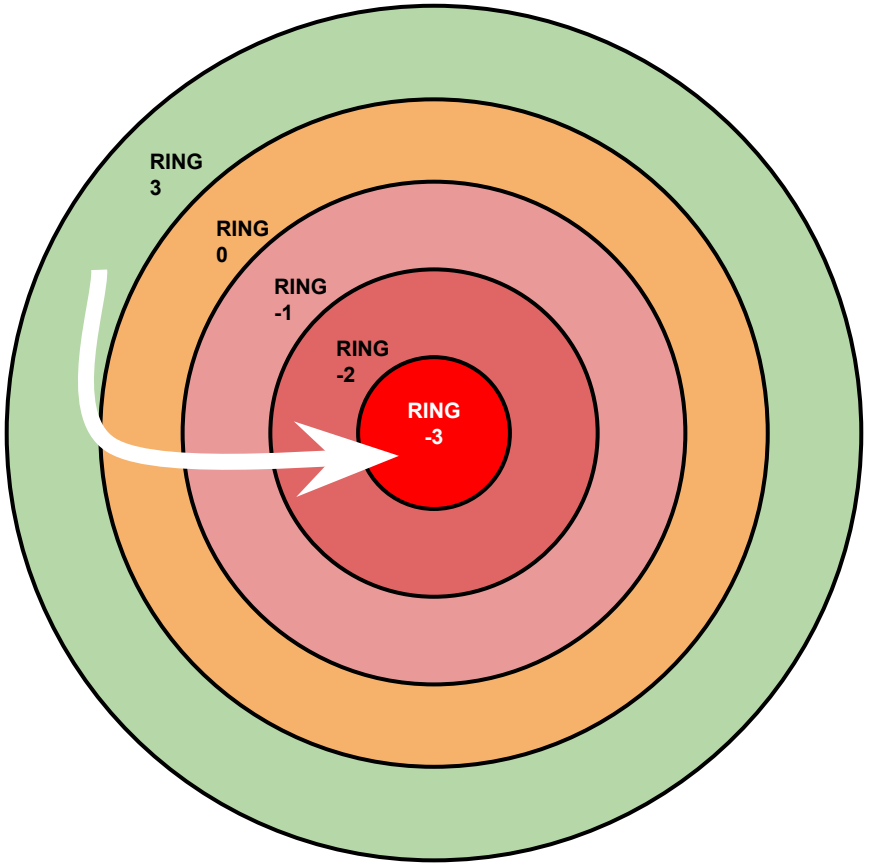
- *Research on pulling off / preventing "priv-escs"*
- *Goes beyond to -1, -2, -3*



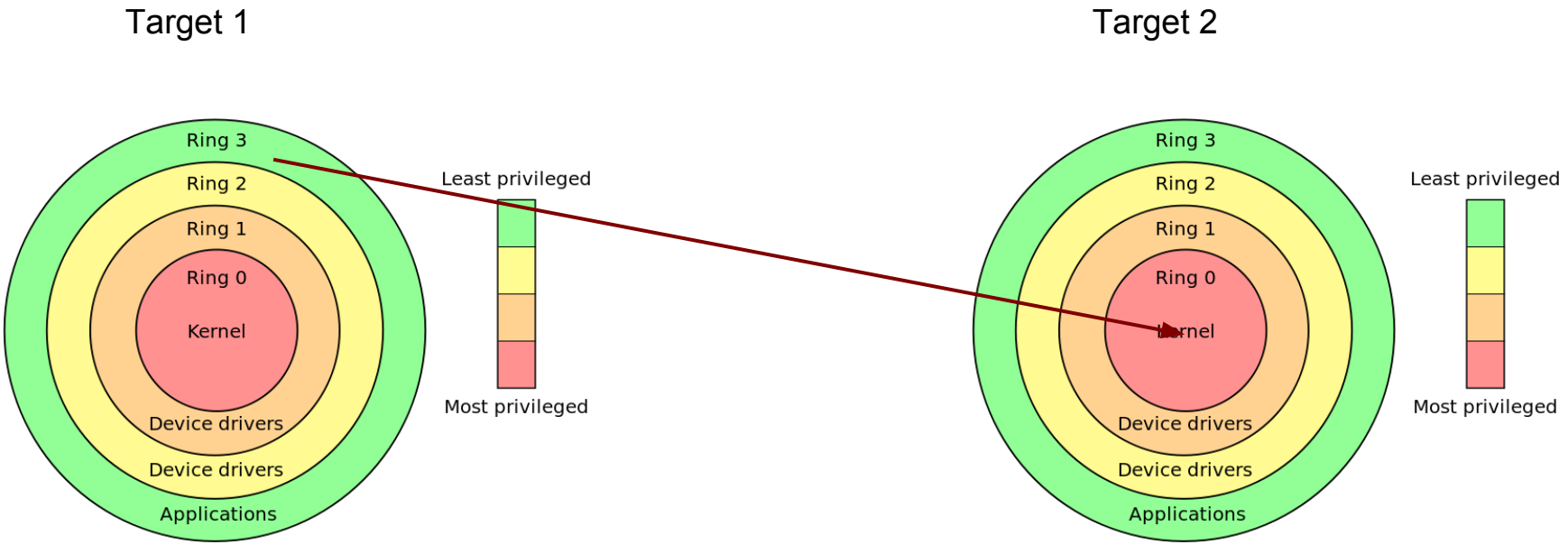
Vulnerability Research

Defense in depth:

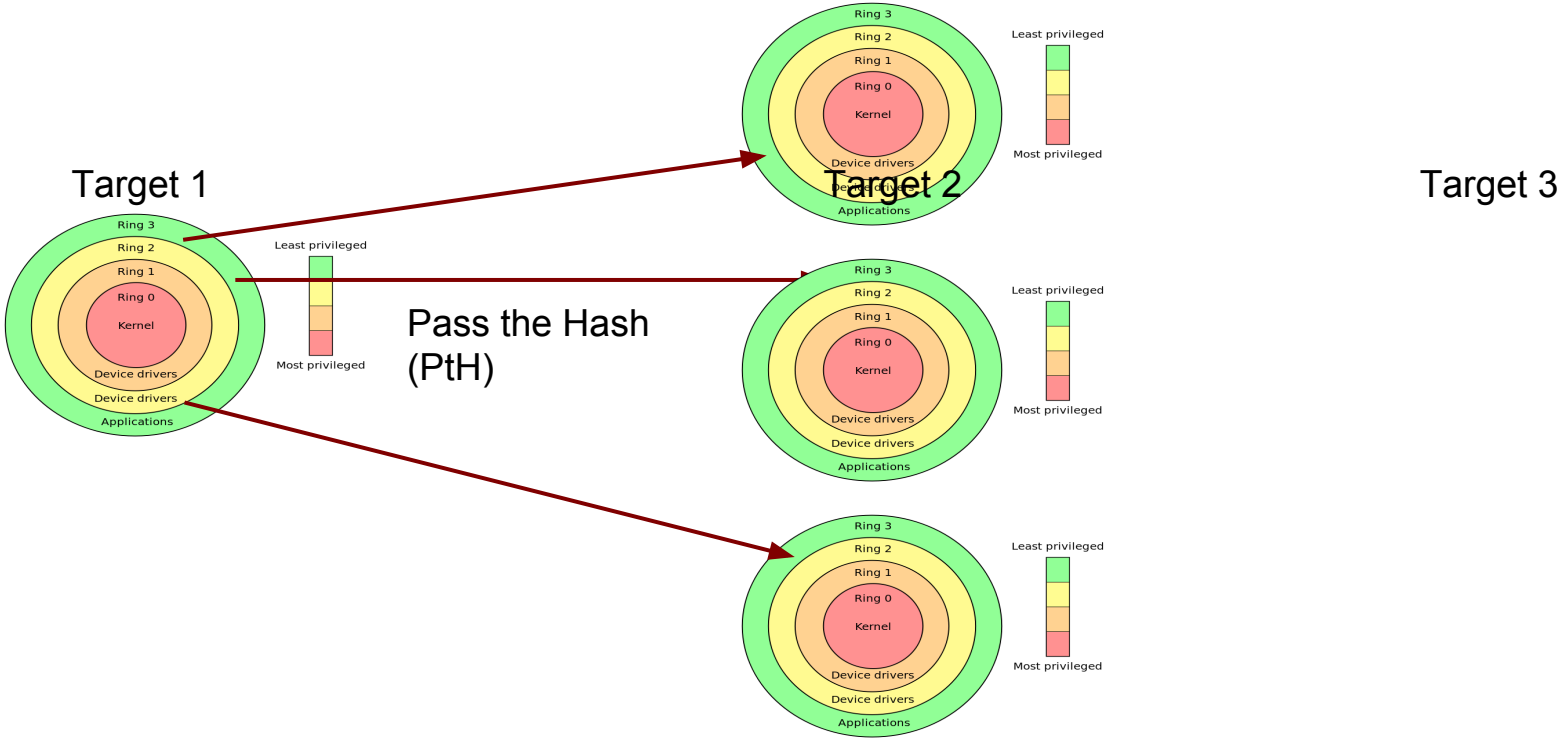
- *In theory attacking something with layered, additive defenses should be harder*
- *Sometimes going from 3 to -3 is harder than 3 to 0*
  - *IPMI (Today's reading)*

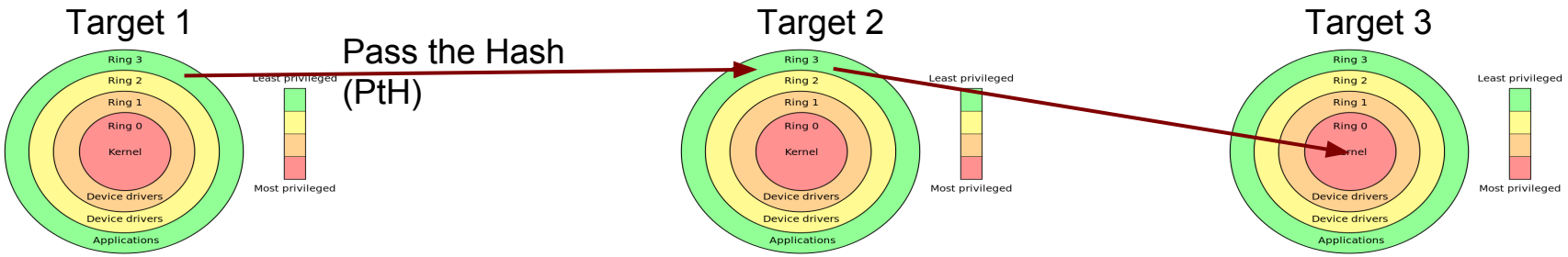


Pivoting: Priv Esc to remote system



Pivoting: Lateral Movement




Target 1
Ring 3
Ring 2
Ring 1
Ring 0
Kernel
Device drivers
Device drivers
Applications
Least privileged
Most privileged
Pass the Hash
(PtH)
Target 2
Ring 3
Ring 2
Ring 1
Ring 0
Kernel
Device drivers
Device drivers
Applications
Least privileged
Most privileged
Target 3
Ring 3
Ring 2
Ring 1
Ring 0
Device drivers
Device drivers
Applications
Least privileged
Most privileged

Linux Topics

Kernel space

- Where most device drivers run (@ ring 1-2)
  - Note that modern micro-kernel trends are pushing drivers into userspace.
- Much different from user space
  - A crash here can be fatal
  - random number generation is very difficult
  - mistakes are unforgiving

Kernel modification

Modifying the kernel requires recompiling it, and rebooting from the new kernel to use it

- unless ksplice or kexec are used

Mistakes are fatal

Difficult to use this in attack chain

init process (user space, ring 3)

- Init is the father of all processes
- it establishes and operates the entire user space
- takes a parameter: runlevel (from 1 to 6)
  - run level determines which subsystems are run
- Executes:
  - scripts to set up all non-operating system services and structures for the user environment
    - checks and mounts the file system
    -
  - spawns the gui (if configured to)
- Then presents the user with the login screen

init process (user space), cont

- init scripts are located usually in directories such as:
  - /etc/rc...../
- The toplevel configuration file for init is at /etc/inittab
- Init checks for the runlevel parameter in /etc/inittab as well.

init process (user space), cont..

init goes dormant after all of the specified processes have been spawned

- waits for 3 things to happen:
  - a process init has started is ending / dying
  - a power failure signal
  - or a request to /sbin/telnit to change the run level

There are other init alternative binaries (depending on the system), such as *systemd*, or *upstart*

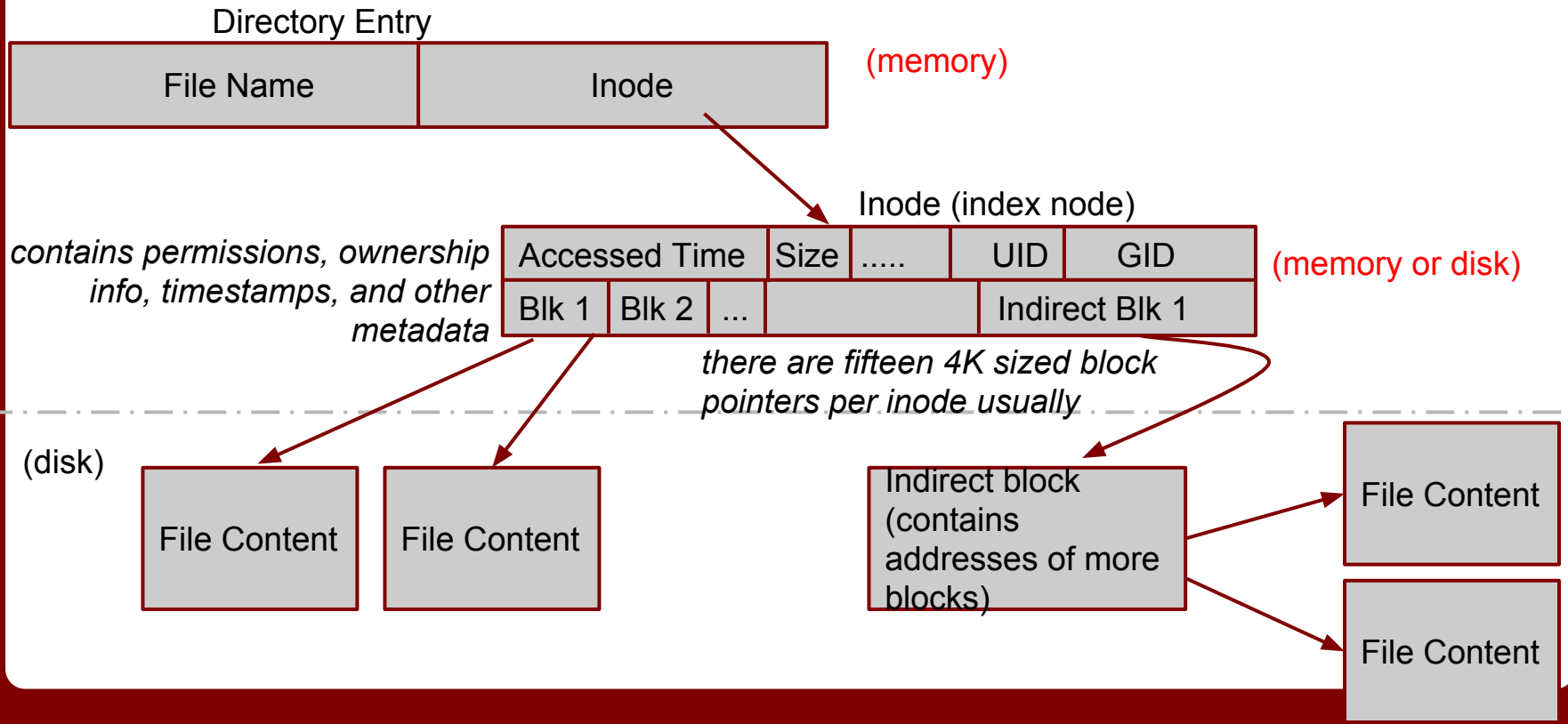
User space

- More security layers here.
- root is king
- common to have a single user account per service (apache's httpd, mysqld)
  - can be set to no login
  - The least privilege principle...

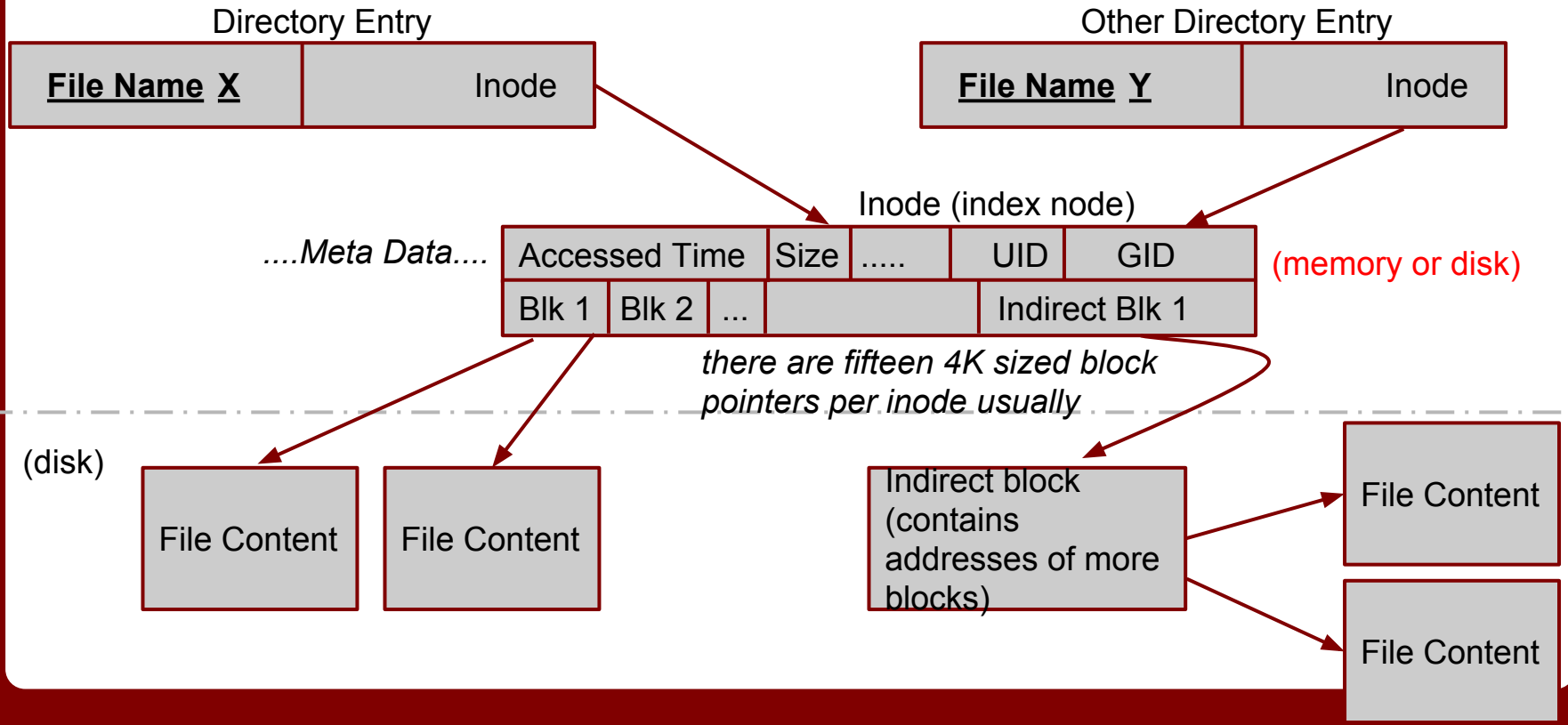
File system, deleting files

- Deleting files can work somewhat like this (on hard disk).
- Data gets left on the disk, and the inode is just unlinked,
- so sectors usually have old data from other files in them->
- **unless securely deleted**

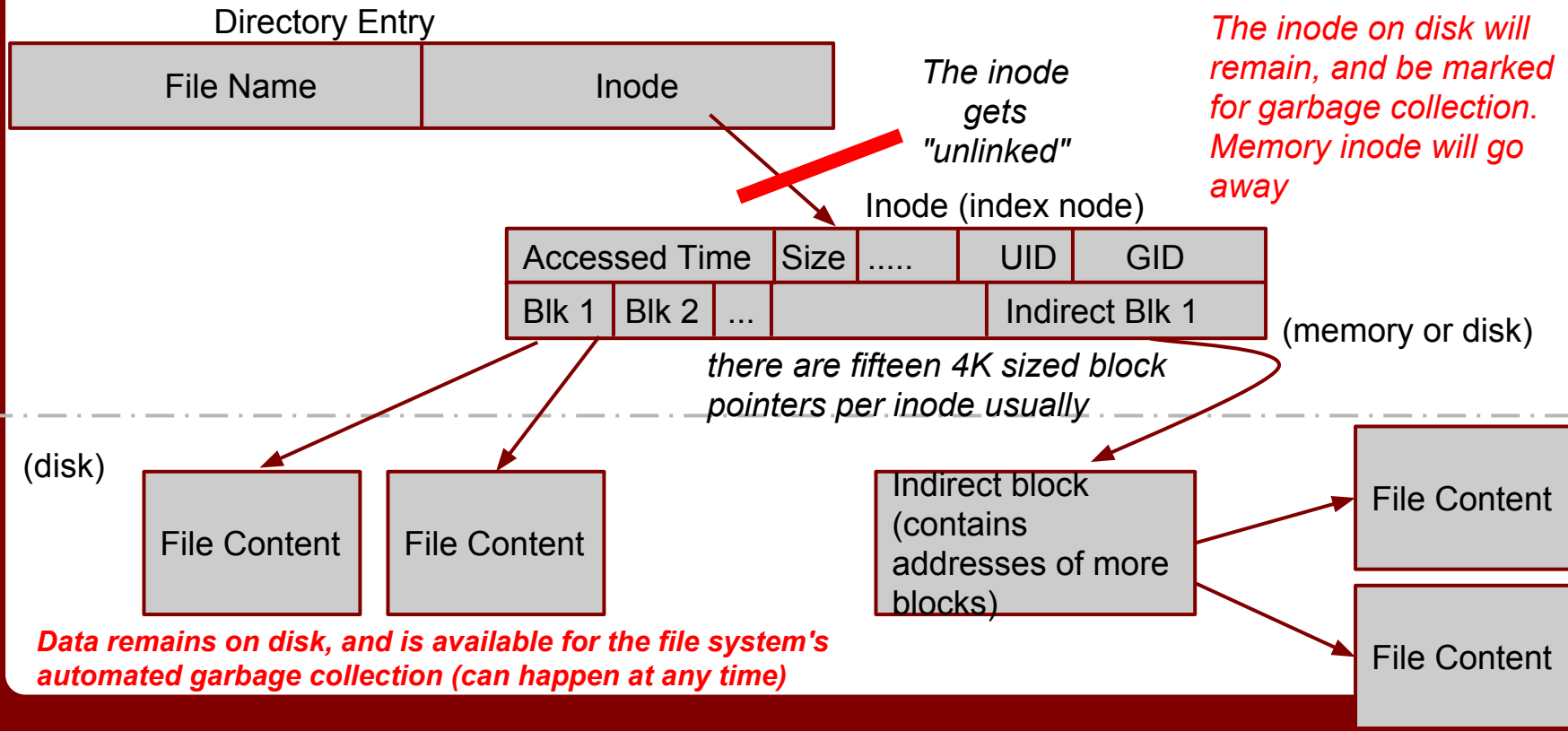
File system basics (ext2/ext3.. ufs, ffs, and others derived from the original fast file system (ffs)



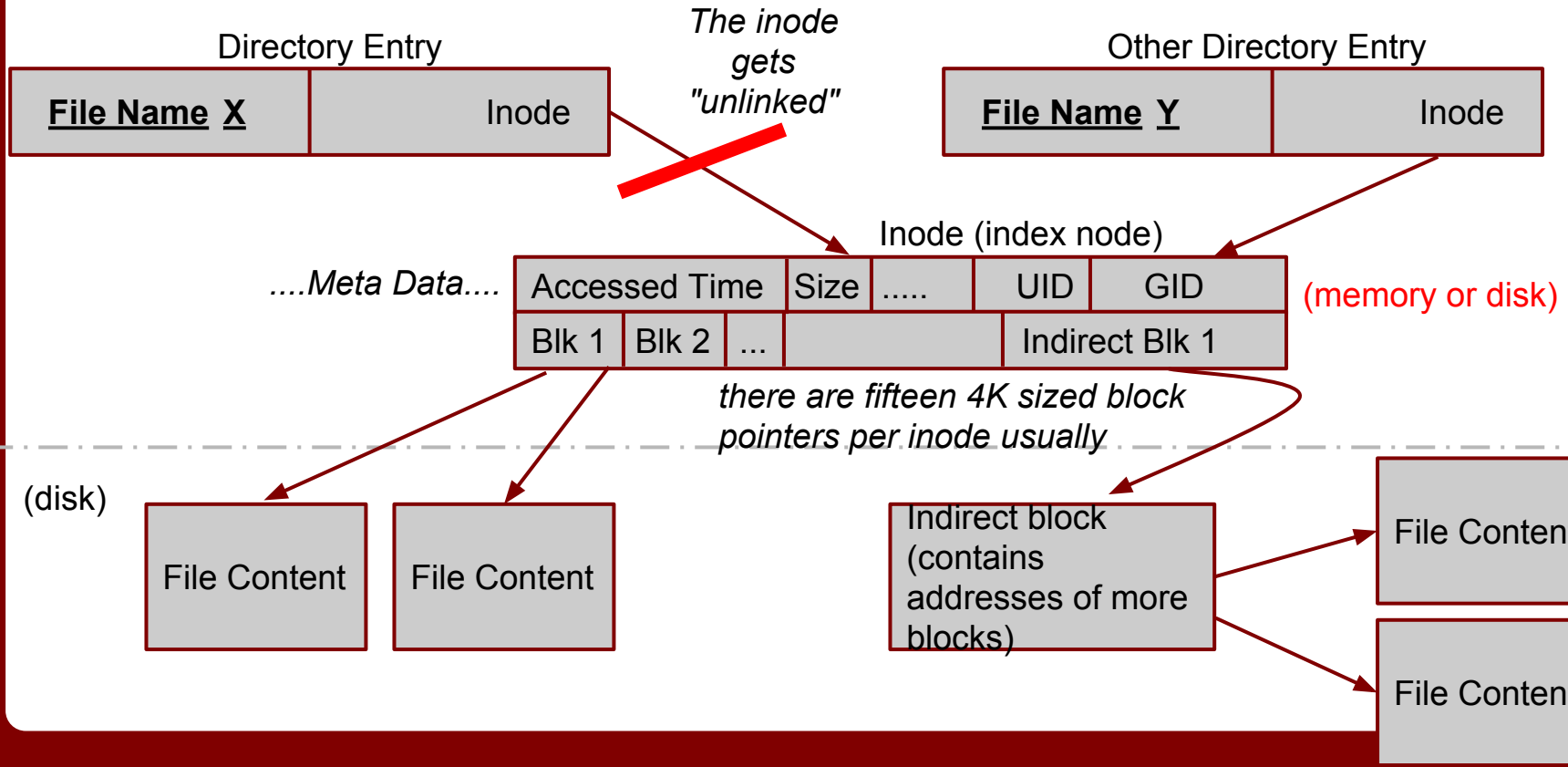
File system basics (ext2/ext3.. ufs, ffs, and others derived from the original fast file system (ffs)



General delete behavior (can differ per file system!!!)



General delete behavior (can differ per file system!!!)



Generic directory layout for linux distros

- Can vary, and can be confusion per distro
- Is not really standardized
- Can be difficult at first to figure out where programs, icons, config files, and other stuff is
- So lets go over the most interesting places:

Generic directory layout highlights

< / >

This is the root directory. This is where the Linux FS begins. Every other directory is under it. Do not confuse it with the root account, or the root account's home directory.

< /etc >

The config files for the Linux system. Most are text files and can be edited by hand

Generic directory layout highlights

< /bin > and < /usr/bin >

These directories contain most of the binaries (programs) for the system. The /bin directory contains the most important programs:

- shells,
- ls, grep

/usr/bin contains other applications for the users

No real clear distinction purpose-wise between the two.

Generic directory layout highlights

< /sbin > and < /usr/sbin >

Most system administration programs are here

< /usr >

Most user applications, their source code, pictures, docs, and other config files. /usr is the largest directory on a linux system.

< /lib>

The shared libraries (shared objects) for programs that are dynamically linked are stored here

Generic directory layout highlights

< /boot >

Boot info is stored here. The linux kernel is also kept here. The file ***vmlinuz*** is the kernel.

< /home >

Where all the users' home directories are. Every user has a directory under /home, and its usually where they store all their files.

Generic directory layout highlights

< /root >

The superuser's (root) home directory

< /var >

contains frequently changed variable data when the system is running. Also contains logs (/var/log), mail (/var/mail), and print info (/var/spool)

< /tmp >

scratch space for temporary files

Generic directory layout highlights

< /dev >

- Contains all device info for the linux system.
- Devices are treated like files in linux, and you can read/write to them just like files (for the most part).

< /mnt >

- Used for mount points
- HD's, USB's, CD-ROM's must be mounted to some directory in the FS tree before being used. Debian sometimes uses /cdrom instead of /mnt

Generic directory layout highlights

< /proc >

- This is a special, and interesting directory.
- /proc is actually a virtual directory, b/c it does not actually exist.
- It contains info on:
  - the kernel
  - all processes info
- Contains special files that permit access to the current configuration of the system.

Users and groups

File permissions are specified in terms of the permissions of

1. The file owner (self)
2. The file's group members (group / business)
3. and everyone else (other)

ls -l

chmod

- -R recursive, ie include objects in subdirectories
- -f force, forge ahead with all objects even if errors occur
- -v verbose, show objects processed

Users and groups

chattr / lsattr

chown

chgrp

/etc/passwd and /etc/shadow

The /etc/passwd file contains users':

- user id
- password hash or info (usually not included in the etc/passwd file anymore...)
- user identifier
- group identifier *(also stored in /etc/group)
- the user's home directory path
- the program that launches when the user logs in (i.e. their shell)

entries look like:

```
jsmith:x:1001:1000:Joe Smith,Room 1007,(234)555-8910,(234)555-0044,email:/home/jsmith:/bin/sh
```

/etc/shadow

man 3 crypt (shows the encryption man page)

- options are:
  - 1: it uses MD5.
  - 5: it uses SHA-256.
  - 6: it uses SHA-512.
  - $2a$: it uses blowfish, not supported everywhere.
  - Otherwise it uses DES.
- md5 no longer secure
- default is DES based, with a 2 character salt string [a–zA–Z0–9./]
  - salt is used to perturb the hash, in one of 4096 ways

The least privilege principle

every process/user/program must be able to access **only the information and resources** that are necessary for its legitimate purpose

- i.e. single no-login user accounts for services like httpd, msqld, etc...
- jails
- security in depth
- Makes logs cleaner

chroot

The root directory (i.e. /) is stored within each process's entry in the **process table**. All the chroot() system call does is to change the location of the root directory for that process.

- Is seen as an OS-level virtualization mechanism.
- The result is called a "chroot jail"
  - can be easily broken. Nothing prevents a program from chrooting out of the jail typically...

Existing OS level virtualizations

Mechanism	Operating system	License	Available since/between	Features								
				File system isolation	Copy on Write	Disk quotas	I/O rate limiting	Memory limits	CPU quotas	Network isolation	Partition checkpointing and live migration	Root privilege isolation
chroot	most UNIX-like operating systems	Proprietary BSD GNU GPL CDDL	1982	Partial[1]	No	No	No	No	No	No	No	No
iCore Virtual Accounts	Windows XP	Proprietary/Freeware	2008	Yes	No	Yes	No	No	No	No	No	?
Linux-VServer (security context)	Linux	GNU GPL v.2	2001	Yes	Yes	Yes	Yes [2]	Yes	Yes	Partial[3]	No	Partial[4]
LXC	Linux	GNU GPL v.2	2008	Partial[5]	Partial. Yes with Btrfs.	Partial. Yes with LVM or Disk quota.	Yes	Yes	Yes	Yes	No	No[6][7][8]
OpenVZ	Linux	GNU GPL v.2	2005	Yes	No	Yes	Yes [9]	Yes	Yes	Yes[10]	Yes	Yes[11]
Parallels Virtuozzo Containers	Linux, Windows	Proprietary	2001	Yes	Yes	Yes	Yes [12]	Yes	Yes	Yes[10]	Yes	Yes
Container/Zone	Solaris and OpenSolaris	CDDL	2005	Yes	Partial. Yes with ZFS	Yes	Partial. Yes with Illumos. [13]	Yes	Yes	Yes[14]	No[15]	Yes[16]
FreeBSD Jail	FreeBSD	BSD	1998	Yes	Yes (ZFS)	Yes [17]	No	Yes [18]	Yes	Yes	No	Partial[19]
sysjail	OpenBSD, NetBSD	BSD	- no longer supported as of 03-03-2009	Yes	No	No	No	No	No	Yes	No	?
WPARs	AIX	Proprietary	2007	Yes	No	Yes	Yes	Yes	Yes	Yes[20]	Yes[21]	?
HP-UX Containers (SRP)	HPUX	Proprietary	2007	Yes	No	Partial. Yes with logical volumes	Yes	Yes	Yes	Yes	Yes	?
Sandboxie	Windows	Proprietary/Shareware	2004	Yes	Yes	No	?	?	?	?	?	?

source: http://en.wikipedia.org/wiki/Jail_(computer_security)

File ownage and setuid setgid

Set User Id (upon execution) and Set Group ID (upon execution).

- Are flags that allow a binary to be executed with the permissions of the binary's owner

Certain applications need to execute under other user account permissions

- example: `passwd`
  - modified the `/etc/passwd` or `/etc/shadow` files which are owned by root.
    - Can't just let anyone edit this freely

Process Permissions: Euid vs Ruid

Effective User ID = UID number

- Determines permissions of process

Real User ID = username

See http://www.cyberciti.biz/tips/linux-more-on-user-id-password-and-group-management.html

setuid (as root) programs

- these programs have complete access on a UNIX system
- virtually ***every*** attack chain involves a focus on attacking these programs
  - they are the single points of failure
  - once attackers get any form of access, they want to escalate to root

Some example results (ubuntu12.04)

/usr/sbin/uuidd
/usr/sbin/pppd
/usr/lib/openssh/ssh-keysign
/usr/lib/eject/dmcrypt-get-device
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/pt_chown
/usr/lib/chromium-browser/chromium-browser-sandbox
/usr/bin/lppasswd
/usr/bin/sudo
/usr/bin/passwd
/usr/bin/chfn
/usr/bin/X
/usr/bin/pkexec
/usr/bin/arping
/usr/bin/mtr
/usr/bin/gpasswd

/usr/bin/newgrp
/usr/bin/at
/usr/bin/chsh
/usr/bin/traceroute6.iputils
/usr/bin/sudoedit
/sbin/mount.ecryptfs_private
/bin/su
/bin/ping6
/bin/fusermount
/bin/mount
/bin/ping
/bin/umount

This can all be seen as the attack surface for any suid permission escalation attacks

good source:
http://www.acm.uiuc.edu/workshops/security/setuid.html

Priv Esc Attack Surface

- suid priv escs
- kernel priv escs
- daemon exploits / root process exploits
- weak passwords
- …

setuid and setgid on directories

- entirely different
- setuid in a directory:
  - does nothing! Is disabled on almost all unix systems
- setgid on a directory:
  - new files and new subdirectories within inherit the directory's gid
    - instead of the creator's primary gid
    - enables shared workspaces for groups

Access Control Lists

- Usually are disabled by default
- Extendeds the owner/group/other access model to allow much finer control
  - can specify permissions for each individual user and group defined in the system
  - is a *mount* option that can be turned on for specific permissions in the /etc/fstab file
  - example entry in /etc/fstab:

```
UID=dfb1151a-be85-496f-8c3c-32804b947446 /       ext4    errors=remount-ro,acl 0       1
```

Access Control Lists

On debian can then install ACL utilities

$ apt-get install acl

or use eiciel (a GUI based ACL manager)

Example use:
Say we have a file "target.txt" that we want the following to be able to edit:

- joe (the CEO)
- developers-g (the developer's group)
- qa-g (the quality assurance & testing group)

ACL Example

developers-g is the group owner for Target.txt

To enable ACL:

```
$ setfacl -m group:developers-g:rw- Target.txt
```

TO enable R/W perm for qa-g, and joe:

```
$setfacl -m group:qa-g:--x,user:joe:rw- Target.txt
```

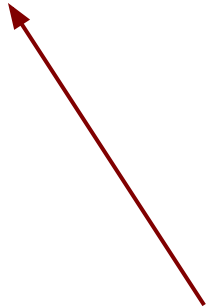
ACL Example

Pretty simple!!

Note:

ls- l will not display the ACL of a file, only if an ACL is enabled by a **+** sign at the end of permissions. Example:

-rw-rw-r--**+** 1 bob bob 60097 2013-01-01 10:55 Target.txt



Extended file attributes (xattr)

supported by ext2, ext3, ext4, JFS, ReiserFS, XFS, Btrfs...

attr / lsattr / chattr interesting uses:

- chattr +i = immutable (means no one, not even root can change/delete/link the file)
- chattr +a = make file append-only (great for logs security!)
- chattr +s = secure deletion for file

By default only root can use xattr, but can be enabled for all in /etc/fstab with the user_xattr option

tty

A TTY is a native terminal device (the backend is either kernel or hardware emulated)

- named after TeleTYpewriter (TTY)
- TTY ports are direct connections to the computer
  - i.e. keyboard, mouse, or serial connection

- commands:
  - who - lists all users and their terminals
  - chvt - switch to another terminal (requires root)
- no GUI (i.e. not the X environment)

pts

Pseudo Terminal Slave

- Is a terminal emulated by another program
  - xterm, screen, ssh, expect, GNOME terminal, Mac OSX terminal....
  - is created through
- Can switch terminals:
  - screen windowlist

Daemons

- view them via "service" command
- background processes
  - syslogd (system logging process)
  - sshd (ssh daemon for incoming ssh connections)
- almost all daemons are spawned by init
- standard behavior:
  - have no tty (controlling terminal)
    - ignore all terminal signals except SIGTERM
  - have no open file descriptors (no I/O)
  - dissassociated from process group

Kernel Modules

a Loadable Kernel Module (LKM) is an object file that contains code that extends the running kernel.

- typically used to add support for new hardware, file systems, or adding system calls...
- Convenient method for modifying the kernel
  - can be abused by attackers though...

cronjobs

cron is the time-based job schedule in Unix systems.

cron enables users to schedule jobs (commands or shell scripts) to run periodically or at certain times or dates

Commonly used to automate system maintenance or administration

Command history

can view previous commands:

```
history
```

can repeat previous commands:

```
!!
!ssh
```

can print the command instead of repeating it:

```
!ssh:p
```

logs

stored typically in /var/log/

- messages - General log messages
- **auth.log, faillog, lastlog** -authentication logs
- boot.log - System boot log
- syslog (a log file and a C command)
- daemon.log - running services such as ntpd, httpd, and etc logs
- kern.log - kernel log file
- mysql.log, mysql.err - database logs (different if postgres, mongo, or other DB)

logs continued

- user.log (user level/application logs)
- There are also virtual machine logs too
- /var/log/apache2/ for apache2 logs
- /var/log/snort/ for snort logs
- and more....

There is a gui based log viewer for GNOME:

`gnome-system-log`

logs are important also for attackers

- brute force (i.e. ssh) attempts leave a big footprint
- remote logins (ips/domains are logged)
- system modification
- kernel modification
- module loading / unloading
- daemon logs

logs are important also for attackers

- Firewall / Gateway / Traffic logs
  - IDS
  - IPS
- AD / LDAP / SCP / … logs
- ...

Linux Firewalls (host based)

Modern Linux host-based firewalls rely on a kernel-based system called "netfilter"

- the iptables user-space tool allows for managing netfilter

Netfilter:

- is a framework within the linux kernel
- has 3 standard tables: filter, nat, mangle
- is primarily a connection-tracking system
- does NOT filter traffic itself, it provides functions for other tools to do that

Netfilter framework tables

1. Filter - default and most basic
   - INPUT, OUTPUT, FORWARD chains
   - responsible for system protection
2. NAT
   - Network Address Translation
   - MASQUERADE usage allows for routing multiple private IP addr's through a single public IP addr.
3. Mangle
   - For changing certain packet fields prior to local delivery.

iptables

part of almost all linux distros since 2.4 kernel (2001)

- tables, chains, matches, and targets
- policies are built from an ordered set of rules
- each rule is applied to a chain within a table
- an iptables chain is a collection of rules that are compared, in order, against packets that share common characteristics (inbound packets vs outbound for instance.

iptables

Tables:

- this is a construct that delineates a broad category of functionality
  - packet filtering, NAT, etc.
- 4 tables:

  1) filter (for filtering rules)

  2) nat (for nat rules)

  3) mangle (for specialized rules)

  4) raw (for rules that function independent of the Netfilter connection tracking subsystem)

iptables

Chains:

- Each table has own set of default chains
- The most important built-in chains for us are the INPUT, OUTPUT, and FORWARD chains in the filter table
  - The INPUT chain used for packets destined for the local Linux system after routing calculations are done in kernel *(i.e. addressed to local socket)*
  - The OUTPUT chain is used for packets leaving the system
  - FORWARD chain governs packets routed through the system

iptables

Important **chains** in the NAT table:

- PREROUTING
- POSTROUTING
  - These are used to modify packet headers before and after IP routing calculations are made in kernel

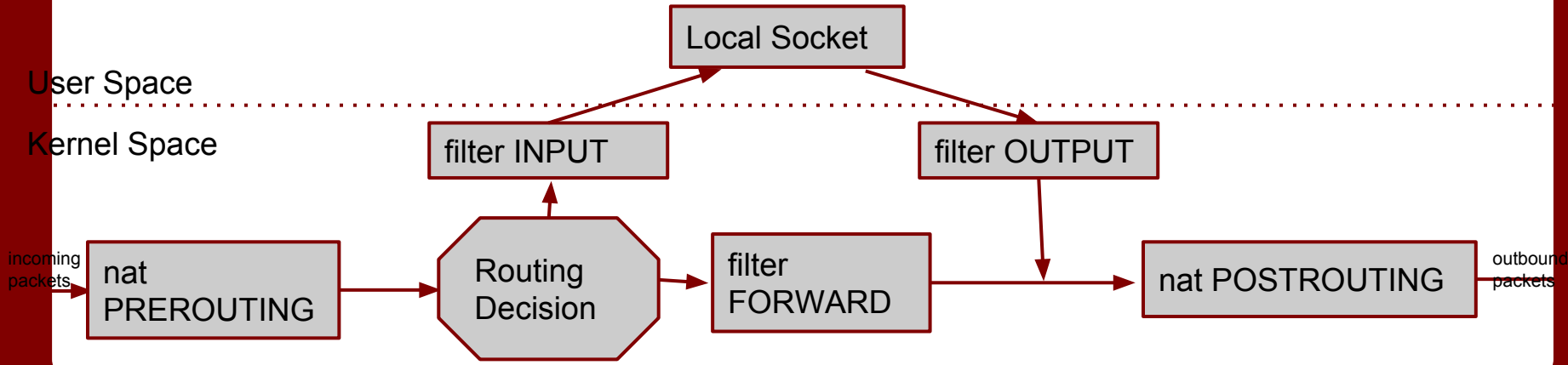


figure source: Linux Firewalls, by Michael Rash

iptables

rule **Matches**:

- Each rule has a set of matches, along with a target
- Target tells iptables what to do with a packet that matches the rule

iptables

match examples

- --source (-s) *Match on source IP or network*
- --destination (-d) *Match on dest IP or network*
- --protocol (-p) *match on an IP value*
- --in-interface (-i) *match on input interface (eth0)*
- --out-interface (-o) *match on output interface (eth0)*
- --state *match on a set of connection states*
- --string *match on a sequence of application layer data bytes (not*

iptables

Targets:

when a packet matches a rule, the target (action) is triggered:

- ACCEPT
- DROP
- LOG (logs a packet to syslog)
- REJECT
- RETURN (continues processing a packet within the calling chain)

iptables Policy Requirements

should be able to initiate the following through to firewall, to outside servers:

- Domain Name System (DNS) queries
- File Transfer Protocol (FTP) transfers
- Network Time Protocol (NTP) queries
- Secure SHell (SSH) queries
- Simple Mail Transfer Protocol (SMTP) sessions
- Web Sessions over HTTP/HTTPS
- WHOIS queries

iptables Policy Requirements

Sessions initiated from the internal network should be statefully tracked by iptables

- packets that do not conform to a valid state should be logged, and dropped
- firewall should be configured with a default *log and drop* policy, to guard against any stray packets, port scans or unallowed connection attempts

ports and services

The /etc/services file contains network port names and numbers which can be used to determine firewall rules

What can go wrong if an attacker gets root

Everything

Questions?

redwood@cs.fsu.edu