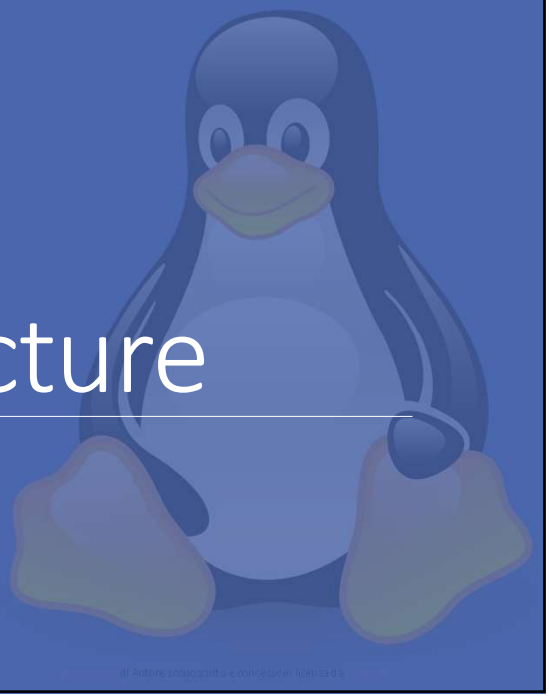
Questionari di Valutazione della didattica

https://esami.unipi.it

1

Linux architecture



2

Learning objectives

Linux has evolved into one of the most popular and versatile operating systems

many features mean broad attack surface

can create highly secure Linux systems

will review:

- Linux structure
- discretionary access controls
- typical vulnerabilities and exploits in Linux
- best practices for mitigating those threats
- new improvements to Linux security model

3

Brief history

Linux is a free operating system based on the UNIX model

First version of the kernel in 1991:

- Released as open source
- Initially for the PC HW platform

Soon complemented with the Free Software Foundation's GNU and other tools from Free BSD, MIT X Window system among others.

Nowadays:

- One of the most advanced and used OS
- Compatible with several HW platforms
- Supports several file systems
- Compatible with lots of existing Unix software (Posix compliant)

4

Linux

distributed under the GNU General Public License (GPL)

- terms set by the Free Software Foundation

it's not public domain, but license gives rights to:

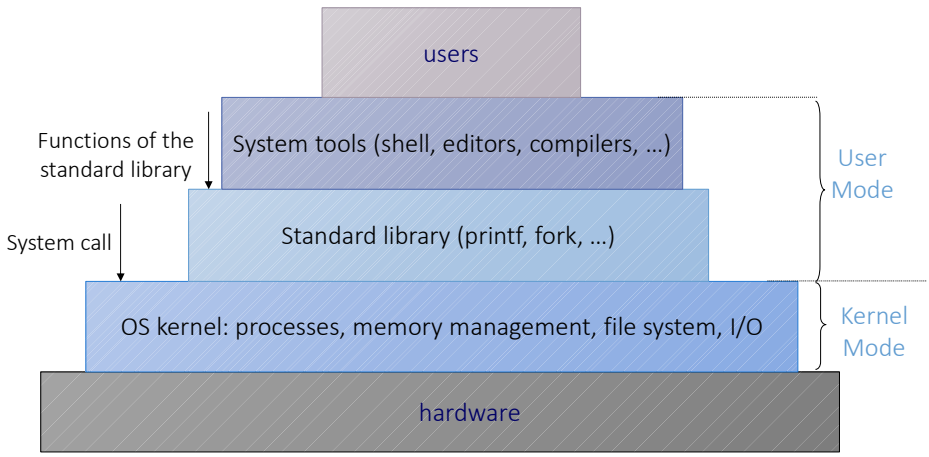
- create a derivative of Linux (but not proprietary)
- Sell distributions (but source code should always be included)

this gave rise to several distributions:

- Precompiled set of Linux packages
- Including kernel, tools and applications
- Both for servers and personal use
- Even for embedded systems

common distributions are Ubuntu, Redhat, Debian …

5



High-level Linux architecture

6

Kernel modules

Kernel modules:

- even developed by third parties
- can be loaded (and unloaded) independent of the rest of the kernel
- may be a device driver, a network protocol or even an entire file system

Linux provide support to:

- Load, unload and manage modules
- Register a driver
- Resolve conflicts among modules

7

Kernel modules

Modules management:

- loading of modules into memory and dynamically linking it to the rest of the kernel
- Periodically checks the dynamically loaded modules: if unused → unload

Register a driver:

- Inform the kernel that a new module (driver) is available
- kernel table to keep track of currently loaded modules
- modules in the kernel table can be device drivers, network protocols, file systems or binary formats.

Resolution of conflicts among modules:

- mechanisms to let modules access and share correctly the system resources
- avoids two drivers use the same hardware
- keeps kernel tables of hardware resources allocated to each module
- If the requested resource is not available, the request is rejected

8

Linux processes

Processes in Linux created using the combination of `fork()` end `exec(·)` system calls

A process:

- has a parent (the process that created it). The relationship parent-child give rise to a process hierarchy.
- Inherits from the father the access rights to resources.
- Has three kind of properties:
  - Identity
  - Environment
  - Context

9

Linux processes:

identity

Process ID (PID)

- a unique identifier of the process in the system
- PID used by system and other process to refer the process (to signal, wait etc.)

Credentials

- The process is associated to a user ID and one or more group IDs
- Determine the process's rights to access files and system resources

Personality

- An ID specific for Linux processes!
- Defines a specific execution domain for the process
- For example it determines the behavior of system calls or how to map signals into actions
- Let Linux support the execution of binaries compiled for other Unix-like OS

Namespace

- wraps a global system resource (the file system or a portion of it) in an abstraction
- processes see the namespace as their own isolated instance of the global resource. Other processes cannot see any change applied to this namespace
- Can be used to implement containers
- For example, some processes share the same file system, but each see it as if it was their own file system, with its own root directory and mounted FS

10

Linux processes:

environment

It is inherited from the parent

It is composed by two arrays:

- argument array: contains the arguments of the command line used to run the program (starts with the name of the program itself).
- environment array: contains "NAME=VALUE" pairs that associate named environment variables with strings values.

A simple and effective way to:

- pass information from the parent to the child processes
- pass the customization of the operating system to a child process (SO customization on a per-process base)

11

Process Context

The main context information of a process is the **scheduling context**:

- In the processor when it is executed
- In the process control block (user area) when it is suspended

Also include kernel tables to keep track of the system resources allocated by the process:

- **File table** contains the list of opened files associated to their file descriptors (system resource acquisition in Unix means opening a file)
- **signal-handler array** defines the handlers in the process' s address space to be called when the process receives a specific signal
- **file-system context**: described the current root and default directories
- **virtual-memory context:** describes the contents of the private address space of the process

12

Memory Management

Organizes the physical memory in different zones:

- Zone_DMA (0-16MB) mapped in kernel space (for I/O)
- Zone_NORMAL (16MB-896MB) mapped in kernel space
- Zone_HIGHMEM (896MB-above) can be mapped in kernel or user space:

  In a 32 bits architecture typically the 3 high GB are for user-space processes, kernel uses 1GB low memory

Keeps track of the free physical pages

Manages the virtual memory of the processes:

- Allocation/deallocation of pages or blocks of pages
- Virtual memory (pages swap in/out on secondary memory)
- Mapping files into process address space

13

Virtual Memory Manager

Maintains the address space of each process:

- Allocates virtual pages on demand
- Implement a page replacement policy:
  - Loads pages from disk to the physical memory
  - Swaps out pages no longer in use

Keeps two separate views of the virtual memory of a process:

- The physical view describes the mapping of the virtual pages in the physical pages (or in the backing store on disk)
- The logical view keeps track of the **regions** in the virtual spaces that are allocated (code, stack(s), heap, libraries, memory mapped files, etc.)

14

Virtual Memory Manager

The creation of a process with a `fork()` triggers the allocation and initialization of a new virtual space:

- The kernel initializes the virtual space of the child with a copy of that of the parent
- To this purpose the kernel duplicates the page table (physical view) of the parent …
- … this means that the virtual space of the child is actually shared with that of the parent
- However, the parent and the child may write their memory differently. Hence, to avoid interference:
  - each page is set with the copy-on-write clause
  - the count of reference to each page is incremented
  - Only when the parent of the child modify a page this page is actually duplicated in two different physical pages

Also the execution of an `exec()` system call triggers the creation of a new virtual space:

- This time it is entirely new and initialized with the code from the executable file

15

Virtual Memory Manager

The virtual memory of a process is organized into **regions**:

- contiguous spaces in the virtual memory
- usually associated to a backing store area (usually a file)
- they are associated to management information:
  - if shared with other processes
  - if copy-on-write is active for the region
  - Other protection information (read-only etc…)

16

Kernel Virtual Memory

Linux reserves a region of the virtual address space of every process for its own kernel

The virtual memory area reserved for the kernel contains two regions:

- A static area that contains a page table that gives access to the entire physical memory at the kernel
- An area allocated dynamically that is for the general functions of the kernel (not reserved for specific purposes)

17

Executing and Loading User Programs

Executables can be in ELF and .out binary format

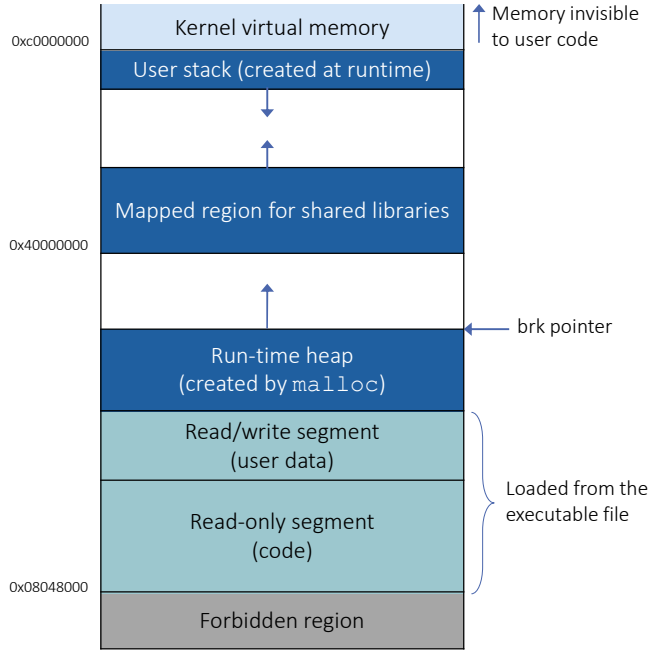
- An ELF-format binary file consists of a header followed by several page-aligned sections
- The ELF loader works by reading the header and mapping the sections of the file into separate regions of virtual memory

The executable file is mapped but not loaded onto the virtual memory

Once in execution the process will refer its memory causing the page faults

In turn, these will cause the kernel to upload its pages in memory

18



Memory Layout for ELF Programs

19

Static and Dynamic Link to Libraries

Two ways to link libraries to user code:

Static: they are embedded directly in the executable file binary code

- The main disadvantage of static linkage is that every program generated must contain copies of exactly the same common system library functions

Dynamic: libraries are linked at run time

- The advantage is that dynamic libraries can be shared by different processes

  They can be loaded in memory once for all

20

Static and Dynamic Link to Libraries

Link of user code to dynamic libraries implemented through a special linker library:

- The program contains a small statically linked function called at startup
- This function maps the link library into memory
- The link library determines the dynamic libraries required by the process and the names of variables and functions needed…
- …then it maps the libraries into a region of the virtual memory …
- … and resolves references to symbols contained in the libraries
- Shared libraries are compiled as **position-independent code** (**PIC**) so can be loaded anywhere

21

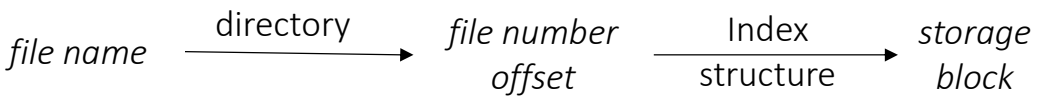
File System

A hierarchical tree of directories (an abstraction implemented by the virtual file system – VFS)

The VFS defines the following structures:

- ***i-node object*** *to represent an individual file, grouped in the* ***i-list***
- ***file object*** *to represent an open file*
- ***superblock object*** *to represent an entire file system*
- ***dentry object*** *to represent an individual directory entry*

22



The index structure is the i-list, an array of i-nodes

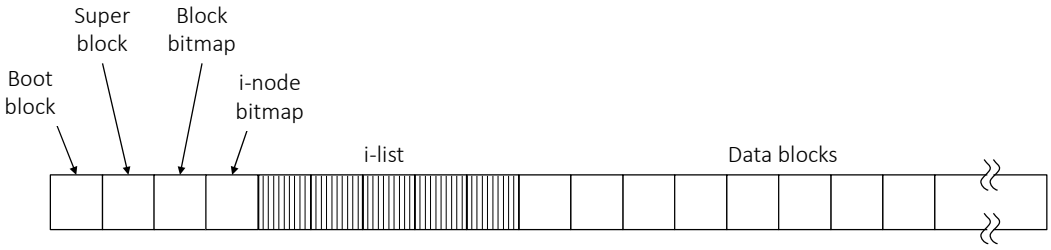
File number offset is the number of the i-node in the i-list

Each i-node:

- Metadata
  - *File owner, access permissions, access times, …*
- Set of pointers to data blocks

File system logic

23



Physical disk organization

Disk partition organization

24

Superblock :

- 2 sectors (1024 bytes) that describe the file system
  - Volume label (a string)
  - Block size
  - # of blocks per group*
  - # of reserved blocks before the 1st block group
  - Superblock block group number
  - # of free i-nodes and blocks (total all groups)
- For block group 0, the first block is for the boot
- Copies of the superblock are in the first block of each block group

*a disk partition can be further partitioned into block groups (see later)

Disk partition organization

25

It's a fixed-size structure

- In FFS, EXT2, EXT3: 128 bytes. In EXT4: 256 bytes

Contains:

- Metadata
- Array of data block pointers (typically 12)
  With 4KB blocks => max size of 48KB
- Indirect block pointer
  pointer to disk block of data pointers
  4KB block size => 1K pointers to data blocks => 4MB
- Doubly indirect block pointer
  Doubly indirect block => 1K indirect blocks
  4GB (+ 4MB + 48KB)
- Triply indirect block pointer
  Triply indirect block => 1K doubly indirect blocks
  4TB (+ 4GB + 4MB + 48KB)

i-node

26

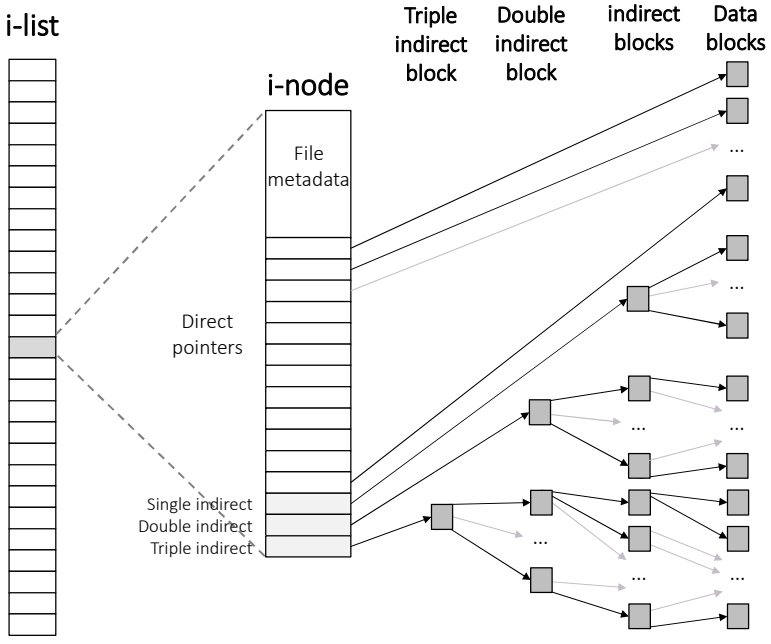
i-nodes

Files metadata in the i-node:

- File type
- Access rights (owner, group, others)
- File length in bytes
- Last access time
- Last i-node change time
- Last data modification time
- Deletion time
- Hard links counter
- Number of data blocks in the file
- File access control list
- Directory access control list

27

i-list and i-node



28

More on i-nodes

File_type	Description
0	Unknown
1	Regular file
2	Directory
3	Character device
4	Block device
5	Named pipe
6	Socket
7	Symbolic link

File type:

- regular files, pipes, etc.
- use data blocks in different ways

Regular File:

- needs data blocks only when it starts to have data
- When first created, empty and need no data blocks

Directories:

- a special kind of file whose data blocks store filenames together with the corresponding i-node numbers
- variable length
- the last name field is a variable length array of up to EXT4_NAME_LEN characters (255).

29

Directory entry (dentry)

A list of records of variable length.

Each record:

Field	Description
i-node	i-node number
rec_len	Length of this entry – implicitly points to the next entry
name_len	Length of filename
file_type	File type
name	filename

To delete a record EXT4 just increases the rec_len of the previous record

- hence the content of the old entry actually remains there, until it is overwritten
- can be useful for deep **forensics**…

30

EXT File System

- Allocation of file blocks by means of i-nodes and i-list
- Block size:
  - depends on the total size of the file system
  - can be either 1, 2, 4 or 8 KB per block
- Cluster allocation policy:
  - keeps data blocks of a file contiguous (as much as possible)
    - *makes accesses to the file more efficient (one disk operation to read several data blocks)*
  - allocates new block to a file in groups instead of individually
- Splits a disk partition into block groups, allocates an entire directory within the same group

31

EXT File System

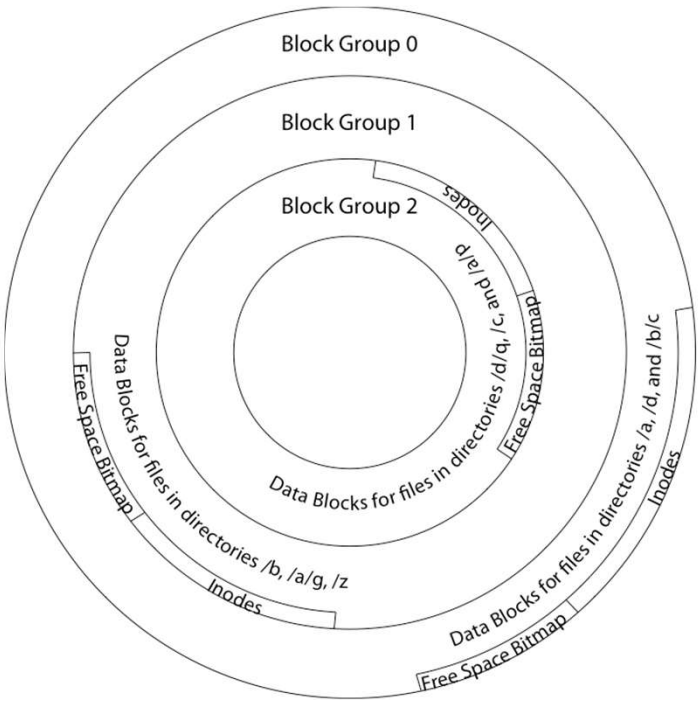
Characteristics of EXT versions

	EXT2	EXT3	EXT4
Introduced in	1993	2001	2006
Max file size	16 GB – 2TB	16 GB – 2TB	16 GB – 16TB
Max FS size	2 TB – 32 TB	2 TB – 32 TB	1 EB
features	Block group, no journaling	Journaling	Extent mapping, multiblock allocation, delayed allocation

Block size	Max file size	Max FS size
1 KB	16 GB	2 TB
2 KB	256 GB	8 TB
4 KB	2 TB	16 TB
8 KB	2 TB	32 TB

Limits of EXT2/EXT3

32

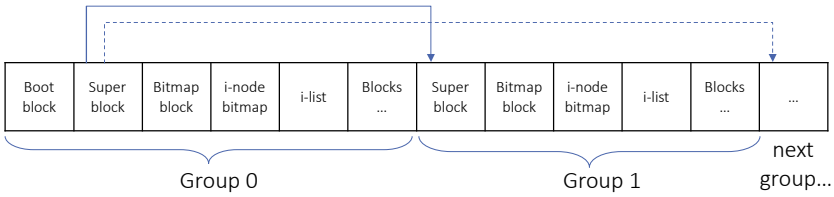


EXT File System

33

Each group has a group descriptor:

- Starting block address of:
  - block bitmap (to keep track of free blocks)
  - i-node bitmap (to keep track of free i-nodes)
  - i-list
- # of free i-nodes& blocks for the group
- Located in the block after the superblock
- Backup copies are in the same block groups as the superblock backups



EXT File System

34

Extents – EXT4

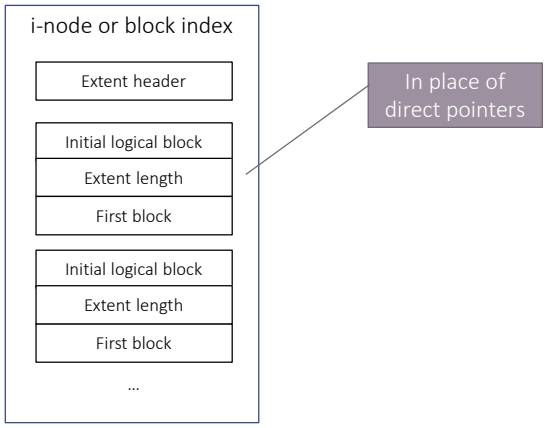
A method to allocate to a file a long sequence of contiguous data blocks

Useful for large files

Compact notation in the block indexes

- length of the extent
- initial block

In files represented with extent changes the list of pointers in the i-node:



36

Journaling – EXT3 & EXT4

- Most often updates to file system not immediate:
  - Updates may remain in main memory before being flushed to disk
- In case of system crash (e.g. power down) the file system may remain inconsistent
  - To overcome this problem, traditional file system made a FS check at each reboot
- Journaling makes the FS more robust and avoids the need of periodic consistency checks
  - To recover from a crash it is sufficient to look in a special disk area (**journal**) that contains the most recent disk write operations

37

Journaling – EXT3 & EXT4

Each update to the file system first written in the journal in the form of a transaction

- Each transaction on the journal has a sequence number

Updates to the file system follow this procedure:

- First write a copy of the blocks to be written in the journal
- When data is committed in the journal then update the file system

38

Journaling – EXT3 & EXT4

When file system crash before a commit to the journal:

- Either the copies of the blocks relative to the high-level change are missing from the journal or they are incomplete;
- Ignore the journal

When file system crash after a commit to the journal:

- The blocks in the journal are valid
- Copy them in the file system

39

Journaling methods:

- DATA:
  - All data and metadata changes are logged into the journal.
  - It's the safest but slowest (requires many additional disk accesses)
- ORDERED:
  - It's the default journaling mode.
  - Only changes to metadata are logged into the journal.
  - Data blocks are written to disk before making any change to the associated metadata
- WRITEBACK
  - Only changes to metadata are logged
  - Data block can be written at any time
  - It is the fastest mode (but not the safest)

Journaling – EXT3 & EXT4

41

ORDERED method (also used in Windows NTFS):

1. **Write data block**: write data to final location; wait for completion
2. **Write metadata in the journal**: write the begin block and metadata to the log; wait for writes to complete.
3. **Journal commit**: Write the transaction commit block to the journal; wait for the write to complete; the transaction (including the data block) is now committed.
4. **Checkpoint metadata**: Write the contents of the metadata update to their final locations within the file system.
5. **Free**: Later, mark the transaction free in the journal.

Journaling – EXT3 & EXT4

42

The proc file system contains a hierarchy of special files that represent the current state of the kernel.

- Files here are virtual
- Still have an i-node
- When accessing these files, proc returns the pertaining data in text form

It is named after its original purpose, which is an interface to the structures within running processes to support debugging tools.

Linux adopted this from Solaris but also added the interface to the kernel.

The proc file system has become quite messy over the years so Linux created the sysfs file system to clean it up.

The /proc file system

43

An example: cat/proc/cpuinfo

```
ste@TabataYoga:/proc$ cat cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 142
model name      : Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz
stepping        : 9
microcode       : 0xffffffff
cpu MHz         : 2904.000
cache size      : 256 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall
nx pdpe1gb rdtscp lm pni pclmulqdq dtes64 monitor ds_cpl vmx est
tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe
popcnt tsc_deadline_timer aes xsave osxsave avx f16c rdrand lahf_lm
abm 3dnowprefetch fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms
invpcid mpx rdseed adx smap clflushopt intel_pt ibrs ibpb stibp ssbd
bogomips        : 5808.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 142
model name      : Intel(R) Core(TM) i7-
7500U CPU @ 2.70GHz
stepping        : 9
microcode       : 0xffffffff
cpu MHz         : 2904.000
cache size      : 256 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags           : ...
```

The /proc file system

44

Linux I/O and IPC

All devices are mapped onto the file system (/dev)

Three classes of devices :

- Block devices allow random access to completely independent, fixed size blocks of data
- Character devices include most other devices; they don't need to support the functionality of regular files
- Network devices are interfaced via the kernel's networking subsystem

Linux provides the classical POSIX Inter-Process Communication (IPC) methods:

- Signals, pipes, etc.

45

Linux Networking

It is a key aspect of Linux

- supports the standard Internet protocols
- It also implements other protocols native to other operating systems (Appletalk, IPX, etc.)

It is implemented by three layers of the kernel:

- The socket interface
- Protocol drivers
- Network device drivers

The protocols include the internet protocol suite:

- implements routing between different hosts anywhere on the network
- UDP, TCP and ICMP protocols are implemented on top of the routing protocol

Linux also filters packets through the firewall management

46

Linux security

Computer Security – Principles and Practice (Pearson, fourth edition)

W. Stallings, L. Brown

47

Linux Security Model

Linux is a direct competitor of Windows
Very often the first choice if you are implementing a server. It is simple and compatible with UNIX (recompatible)

And Unix
Linux's traditional security model is:

- people or processes with "root" privileges can do anything
  - Long limit
- other accounts can do much less

① hence attacker's want to get root privileges

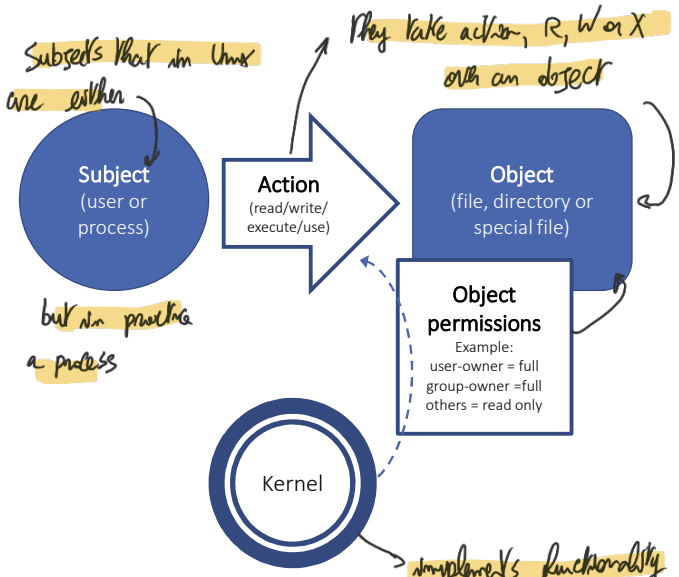
② despite simplicity of the security model, you can run robust, secure Linux systems

crux of problem is use of **Discretionary Access Controls** (DAC)

Unix adopts a DAC, governed by the root user. In windows, because of MAC, administration comes with much more restrictions.

48 This simplifies many things but it's also a weakness :(, but still ②

Linux security transactions



Subjects that in Unix are either

They take action, R, W or X over an object

but in practice a process

implements functionality of Access Control.

DAC here very clear and simple, so you can prove that is free from vulnerabilities.

49

Where does this process take privileges? Exe file comes with owner and group, so this tells your privileges that it might have. ②

Linux security model

A process' program is just an executable file copied into memory

Permissions to the process come into play twice:

- Prior the execution: program file permissions restrict who can execute, access or change it
- During execution: the process runs with the identity of the user and group that executed it ②

Thus the process acts as a user (and a group):

- The kernel will evaluate first that any operation on any object respect the permissions of the user/group on that object

Whoever owns an object can set or change its permissions.

- that's the Linux DAC model's real weakness…
- the system **superuser** account, called "root," can both take ownership and change the permissions of all objects in the system ①
- moreover, it's not uncommon for both processes and administrator-users to routinely run with root privileges, in ways that provide attackers with opportunities to hijack those privileges.

50 DAC decides how to give permission to subjects. By maintaining DAC list related to the subject.

So this is valid, but ① applies.

If sys not hardened enough typical that there are processes running as root.

So those processes are targets for attackers. In Unix there's no MAC access control, unless you use Linux Extras.

File system security

in Linux *everything* as a file

- e.g. memory, device-drivers, named pipes, and other system resources
- hence why filesystem security is so important

I/O to devices is via a "special" file

- e.g. `/dev/cdrom`

also other special files like named pipes

- a conduit between processes / programs

That's why file system security is so important

51

Users and Groups

There are actually two things that are not files…

a user-account (user)

- represents someone capable of using files
- associated both with humans and processes

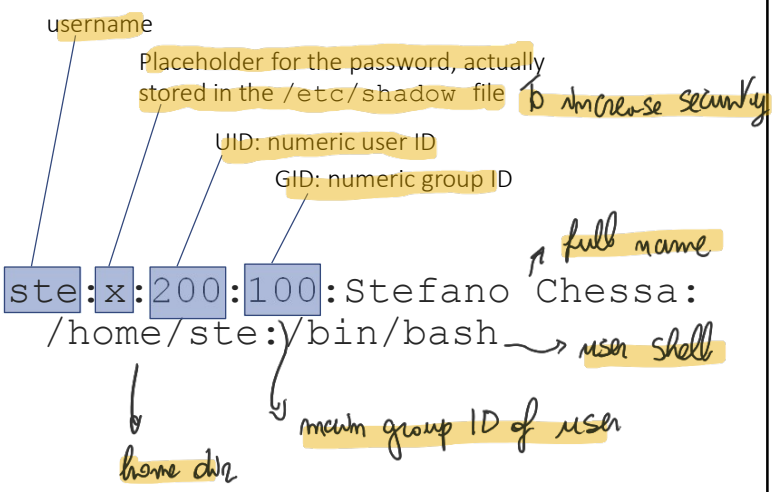
a group-account (group)

- is a list of user-accounts
- users have a main group and may also belong to other groups
- main group membership in the `/etc/passwd` file;
- other memberships in the `/etc/group` file

52

Users and Groups

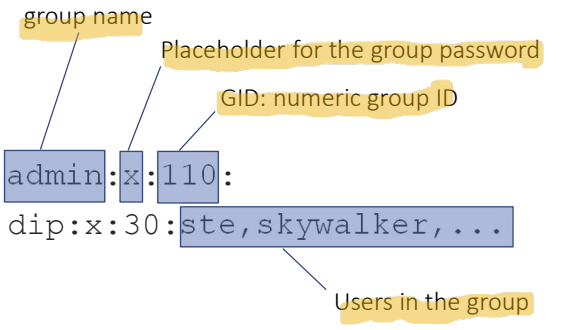
user's details are kept in `/etc/passwd`:



53

Users and Groups

additional group details in `/etc/group`:



can use **useradd**, **usermod**, **userdel** to change users and group memberships

54

File Permissions

files have two owners: a user & a group

each with its own set of permissions…

… and with a third set of permissions for others.

Permissions are to read/write/execute in order user/group/other:

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

Can be set using chmod command

9 bits for access control list in a summed up form

55

Directory Permissions

read = list contents

write = create or delete files in directory

execute = use anything in or change working directory to this directory

e.g.

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

NOTE: what you can do on the file also depends on the directory

56

Directory Permissions

This model is however limited especially for directory for coop purposes

Assume you wish to share a directory with your group's members to let them add files

… you would need to give them the write permission…

… but this would also let them the right to delete!

57

Sticky Bit

had other meaning

originally used to lock file in memory

now used on directories to limit delete

- if set must own file or directory to delete
- other users cannot delete even if have write

set using chmod command with +t flag, e.g.

```
chmod +t shared_dir
```

Note: only applies to that specific directory, not to its child directories

58

Sticky Bit

directory listing includes t or T flag:

a compact representation

```
drwxrwx--T  8  ste dip 288  Mar 25
  01:38 no_delete1
drwxrwx--t  8  ste dip 288  Mar 25
  01:38 no_delete2
drwxrwx---  8  ste dip 288  Mar 25
  01:38 can_delete_dir
```

Note that 'T'/'t' replace now the 'x' right for others.

- 'T' Means that the directory is non executable by others but has the sticky bit set
- 't' Means that the directory is executable by others and has the sticky bit set

59

SetUID and SetGID

setuid bit means program "runs as" file owner
no matter who executes it

setgid bit means run as a member of the group which owns it
again regardless of who executes it

"run as" = "run with same privileges as"

very dangerous if set on file owned by root or other privileged account or group
Use only on executable files, not shell scripts

60

SetUID

setUID set using `chmod` command with `+s` flag, e.g.

```
chmod +s ./own_executable
```

directory listing includes s flag:

```
-rwsrwsrwx  8  ste dip 288  Mar 25
  01:38 own_executable

-rwxrwxrwx  8  ste dip 288  Mar 25
  01:38 norm_executable
```

IMPORTANT: setUID very dangerous, especially if used for root-owned executables

much better to use `sudo` to run an executable as superuser (or any other user)

61

SetGID

setGID set using `chmod` command with `g+s` flag, e.g.

```
chmod g+s ./grp_executable
```

In this way the file is executable with the owner's group privileges

directory listing includes s flag:

```
-rwxrwsrwx  8  ste dip 288  Mar 25
  01:38 grp_executable

-rwsrwsrwx  8  ste dip 288  Mar 25
  01:38 own_executable

-rwxrwx---  8  ste dip 288  Mar 25
  01:38 normal_executable
```

62

SetGid: if you create a file within, every file will inherit the group of the directory

SetGID and Directories

setuid has no effect on directories

setgid has effect. It causes any file created in a directory to inherit the directory's group

- also files created in the future!

useful if users belong to other groups and routinely create files to be shared with other members of those groups

- instead of manually changing its group

63

setuid root vulnerabilities

a **setuid root** program runs as root

- *no matter who executes it*

used to provide unprivileged users with access to privileged resources

- For example to change their passwords

must be very carefully programmed

if exploited due to a software bug, it may allow unprivileged users to wield unauthorized root privileges

distributions now minimise setuid-root programs

system attackers still scan for them!

64

Numeric file permissions

Internally, Linux uses numbers to represent permissions; only user programs display permissions as letters.

The chmod command recognizes both mnemonic permission-modifiers ("u+rwx,go-w") and numeric modes.

A numeric mode consists of four octal digits in this order:

1. special-permissions (4-setUID, 2-setGID, 1 sticky-bit)
2. user-permissions (4-read, 2-write 1-execute)
3. group-permissions (4-read, 2-write 1-execute)
4. other-permissions (4-read, 2-write 1-execute)

The value 0 represents no rights.

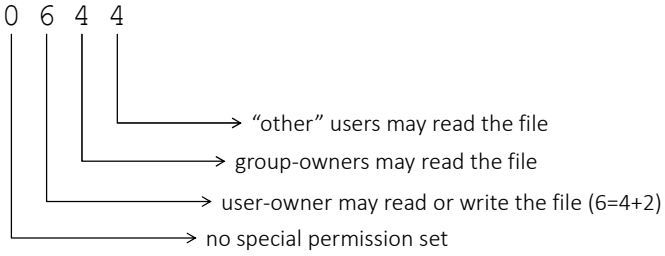
65

Permissions can be encoded in

Numeric file permissions

Example:

```
chmod 0644 mycoolfile
```



66

Access control lists

set an ACL of a given file

Can be configured by using the `setfacl` command.

Grants access rights to specific users or groups

Example, to grants read access to `myfile` to user `ste`:

Modify ACL associated to file

```
setfacl -m u:ste:r myfile
```

entry to apply:

'm' modify, 'x' removes the ACL for the file

'u' applies to a user; 'g' applies to a group

'r', 'w', 'x' the right granted/revoked

1. Flag tells you whether we refer to user or group
2. Name of U or G to which you are applying something
3. The rights.

Access control lists

You can inspect ACL with

Inspecting ACL with `getfacl`

```
1:  # file: mydir/ (it's a dir)
2:  # owner: ste
3:  # group: dipa
4:  # flags: -s-
5:  user::rwx
6:  user:bill:rwx          #effective:r-x
7:  group::rwx             #effective:r-x
8:  group:chess:r-x
9:  mask::r-x
10: other::r-x
11: default:user::rwx
12: default:user:bill:rwx #effective:r-x
13: default:group::r-x
14: default:mask::r-x
15: default:other::---
```

Filename, owner, group

setuid, setgid, sticky

owner rights

owner's group rights

Effective right mask

other's rights

only for directories: default ACL

We have 2 ACL for dirs: 1. Main one. Represents the 9 bits in conventional Unix lists.
And then additional rights to specific users.
* easy way to modify rights for users quickly. Mask will remove privileges

Normal listing will give access right for owner, group and the mask.

The default ACL can be used if you are creating files inside of the directories.

Access control lists

In general, to set an ACL: `setfacl -m rule file`

Where a rule takes the form:
- `u:name:permissions` – set for user `name`
- `g:name:permissions` – set for the group `name`
- `m::permissions` – set the effective mask
- `o::permissions` – set for everybody else

} no name required

To set a default ACL for a directory:
- `setfacl -m d:rule directory`

↳ Rule for default rules.

To remove ACL:
- `setfacl -x rule file` *// removes a specific `rule` from `file`.*
- `setfacl -x u:name file` *// removes permissions of user `name` from `file`.*
- `setfacl -b file` *// removes the entire ACL from `file`*

69

Access control lists

The `setfacl` command can be used on command line or can take an ACL from a file.

API

- among the rights on a file or directory there is that of changing the ACL of a file. Typically granted to the owner and to root.

`setfacl` also gives the possibility to copy an ACL from a file to another

Refer to the manuals for detailed information

70

Protection of OS in memory too, goes beyond AC.

Kernel vs User Space

Users, groups and privileges are only one aspect of Linux DAC…

also memory matters!

Remember:

Kernel space
- refers to memory used by the Linux kernel and its loadable modules (e.g., device drivers)

User space
- refers to memory used by all other processes

Since kernel enforces Linux DAC and security it is critical to isolate it from user
- so kernel-space never swapped to disk
- only root may load and unload kernel modules

Linux implement paging, pages can be swapped on the disk, could be accessible and malfuel.

71 How OS are protecting themselves against hw and drivers? In mac they decide who gets in. For Windows not works. There's a process of certification for drivers and providers. For linux, since it's open source it's on your own risk.

Rootkits

allow attackers to cover their tracks

if successfully installed before detection, all is very nearly lost

originally collections of hacked commands
- hiding attacker's files, directories, processes

now use loadable kernel modules
- intercepting system calls in kernel-space
- hiding attacker from standard commands

may be able to detect with chkrootkit (available at www.chkrootkit.org)

generally have to wipe and rebuild system

72 Methods used by intruders to hide their tracks. Years ago, rootkits were variations of sys commands hiding processes of attackers. Checking integrity of sys commands is important. Rootkit now are kernel module that filter our info by monitoring interrupts.

Linux was less vulnerable than windows, because linux users were more expert. Recent time, with increase of users on linux, sys becomes more vulnerable. Hardening linux system means probing this first of all.

Applications vulnerabilities in Linux: Web

- a very broad category of vulnerabilities
  - because of ubiquity of world wide web have big and visible attack surfaces
- when written in scripting languages
  - not as prone to classic buffer overflows
  - can suffer from poor input-handling
- Nowadays Linux distributions shipped with few "enabled-by-default" web applications
- but users install vulnerable web applications…
- … or write custom web applications having easily-identified and easily-exploited flaws

73

Linux System Hardening

- consider how to mitigate Linux security risks at system and application levels
- first look at OS-level security tools and techniques that protect the entire system

Typically done for specific uses: safe distribution version, update policy for updates etc.

74

OS Installation

security begins with O.S. installation

especially what software is run

- since unused applications liable to be left in default, un-hardened and un-patched state

generally should not run: exclude % of services and applications not required and unsafe.

- X Window system, RPC services, R-services, inetd, SMTP daemons, telnet etc

also have some initial system SW configuration:

- setting root password
- creating a non-root user account
- setting an overall system security level
- enabling a simple host-based firewall policy
- enabling SELinux

config of root and normal users

75

Patch Management

1 cons: in Windows update manager almost automatic. If you are running a critical server, an update could cause problems with services. In those cases, recommended to disable automated update and test on parallel machines before.

installed server applications must be:

- configured securely
- kept up to date with security patches

patching can never win "patch rat-race"

have tools to automatically download and install security updates

- e.g. up2date, YaST, apt-get
- Note: should not run automatic updates on change-controlled systems without testing

76

Network should be hardened. There was a TCP wrapper that was a sort of firewall. Now obsolete. INETD.

Network Access Controls

The network is a key attack vector to secure

TCP wrappers a key tool to check access

- originally tcpd inetd wrapper daemon
  before allowing connection to service checks:
  if requesting host explicitly in hosts.allow is ok
  if requesting host explicitly in hosts.deny is blocked
  if not in either is ok
- also checks on service, source IP, username
- but remember IP addresses can be spoofed…
- now often part of app using libwrappers

77

Network Access Controls

A better (and powerful) tool is the **netfilter** Linux kernel native firewall mechanism

- **iptables** is the **netfilter** user-space front end… **netfilter** often known with this name

Can be used on firewalls, servers, desktops

- Its use if tricky, steep learning curve (especially for common users)
- Do have automated rule generators
- typically for "personal" firewall use will:
  allow incoming requests to specified services
  block all other inbound service requests
  allow all outbound (locally-originating) requests
- if need greater security, use manual config

78

Another method for hardening. Main problem for linux are worms. For viruses, purpose is to propagate. Virus, if sys is protected, is not affecting the sys. Still, anti-v. is useful because malwares are a combination of bugs.

Antivirus Software

- Historically Linux not as vulnerable to viruses
  - more to lesser popularity than security
- System administrators normally patch promptly the system,
  - this is effective for worms, which are the main threat to Linux
- However, patches not sufficient against viruses, which abuse users' privileges
  - non-root users have less scope to exploit
  - but can still consume resources
- Growing Linux popularity mean exploits and more viruses attacks
- Hence antivirus software are becoming more important
  - various commercial and free Linux A/V

79

Config of users is important. Sys admin may need to work for them own specific needs. This might include policy pw definition. There's command for expiration of pw.

User Management

Guiding principles in user-account security:

- need care when setting file / directory permissions
- use groups to differentiate between roles
- use extreme care in granting / using root privs

Remember:

- commands: chmod, useradd/mod/del, groupadd/mod/del, passwd, chage
- info in files /etc/passwd & /etc/group
- manage user's group memberships
- set appropriate password ages (?)

80

SetUid/Gid is Root delegation. This should be configured appropriately. Alternate is sudo.

Root Delegation

have "root can to anything, users do little" issue

"su" command allows users to run as root:

- either root shell or single command
- must supply root password
- means likely too many people know this

SELinux RBAC can limit root authority, but it's complex to configure

An alternative is "sudo":

- allows users to run as root
- but only need their password, not root password
- /etc/sudoers file specifies what commands allowed

The other alternative remains that of configuring user/group permissions

81

Setting up logging is important.
Modern loggers support log for OS, apps even on remote files with encrypted transmission. So intruder can't modify logs.

Logging

Logging is not a proactive control, but it is key to diagnostic and early detection.

Linux logs using syslogd or Syslog-NG

- receive log data from a variety of sources
- sorts by **facility** (category) and **severity**
- writes log messages to local/remote log files

Syslog-NG preferable because it has:

- variety of log-data sources / destinations
- much more flexible "rules engine" to configure
- can log via TCP which can be encrypted

Before use it's a good practice to check and customize the default settings

82

Log Management

How many log files:

- one single file may become cumbersome
- too many files may be problematic as well, need for a balance

Management of log files

- cannot afford to fill the file system!
- must "rotate" log files and delete old copies
- typically use logrotate utility run by cron
- to manage both system and application logs

Linux log systems also support application logging

- must also configure applications

83

Application Security

this is a large topic

many security features are implemented in similar ways across different applications

will review issues such as:

- running as unprivileged user/group
- running in chroot jail
- modularity
- encryption
- logging

84

Running As Unprivileged User/Group

every process "runs as" some user

extremely important this user is not root

- since any bug can compromise entire system

But some processes (e.g. network processes) cannot completely avoid it

- may need root privileges, e.g. bind port
- In these cases the parent may perform the privileged functions as root …
- …and leave the main service to unprivileged child(s)

Better to run different network processes with dedicated user/group

- easier to identify source of log messages

85

Modularity

Applications running as a single, large, multipurpose process can be:

- more difficult to run as an unprivileged user
- harder to locate and fix security bugs in their source code
- harder to disable unnecessary functionalities

Hence modularity is a highly prized feature

- providing a much smaller attack surface
- Examples: postfix vs sendmail, Apache modules, …

86

Sandbox: with chroot you define a portion of FS as if it was the entire FS. You make it appear for users or processes the root dir as that dir. You cannot compromise rest of system.

Chroot jail

- Restricts the server's view of the file system to just a specified portion
- File directories outside the chroot jail aren't visible or reachable
- If a cracker breaches a chrooted application, he must break the jail before making damage to the rest of the system
  - You would need root privileges.
- Main disadvantage is added complexity
- also, some programs do not work under chroot
  - Processes might need to use commands, services of OS that are in different directories. This means making copy of those important dirs, might have work.

Problem in setting it up;

87

Running in chroot jail

Chroot confines a process to a subset of the filesystem, into a specific branch

- maps a virtual "/" to some other directory
- useful if have a daemon that should only access a portion of the file system, e.g. FTP
  - e.g. may map it in `/srv/ftp/public`
- directories outside the chroot jail aren't visible or reachable at all

Contains effects of compromised daemon

Complex to configure and troubleshoot

- must mirror portions of system in chroot jail
- if the process runs as root it can "break out" the jail
- still the advantages are better than the limits

88

Encryption

Sending logins & passwords or application data over networks in clear text exposes them to network eavesdropping attacks

Many network applications now support encryption to protect such data

- often using OpenSSL library

May need own X.509 certificates to use

- can generate/sign using openssl command
- may use commercial/own/free Certification Authorities

89

Logging

Applications can usually be configured to log to any level of detail (debug to none)

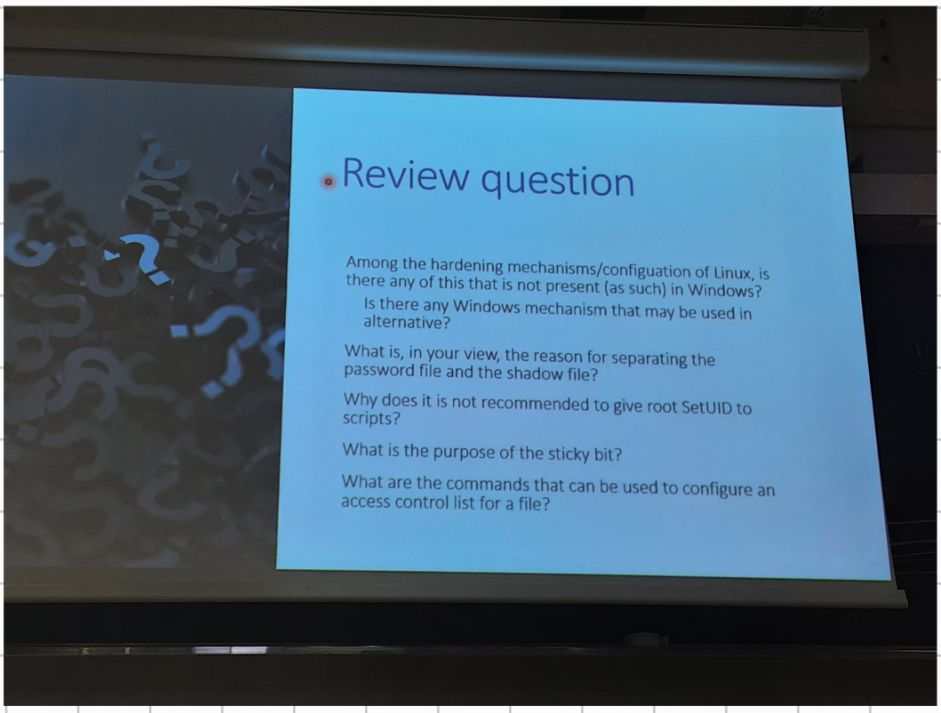
Need appropriate setting

Must decide if use dedicated file or system logging facility (e.g. syslog)

central facility useful for consistent use

Must ensure any log files are rotated

90



MAC in Windows can be used in a similar sandbox way (browser). Not entirely chroot, because you still have access to everything and MAC is mainly thought for integrity, while for chroot you cannot go outside.

2) We can give setUid to everything executable. Binary executables and scripts. Why avoid scripts? There are some attacks on scripts like modifying paths, line separators etc. Simpler to attack them for executables. And binary setuid executables there can usually be developed, are open source, checked by IMs.

Mandatory Access Control

Security-Ehnanced Linux framework

91

Mandatory Access Controls

Mandatory Access Controls (MAC) impose a global security policy on all users

- users may not set controls weaker than policy
- normal admin done with accounts without authority to change the global security policy
- but MAC systems are harder to manage…

Novell's SuSE Linux has AppArmor

RedHat Enterprise Linux has SELinux (Security Enhanced Linux)

"pure" SELinux for high-sensitivity, high-security

92

SELinux is the NSA's powerful implementation of mandatory access controls for Linux

It comes at a cost: it's a complex technology to manage and troubleshoot

The limits of conventional Linux security:

- a race to the last patch
- there's always the possibility of a zero-day vulnerability (hence unpatched)

SELinux does not prevents zero-day attacks, but contains their effects

SELinux

93

For example, suppose we have a daemon called *blinkled:*

- it just reads a text file and blinks out jokes written there in Morse code on the keyboard LED
- runs as the user *Bob*, but it is hijacked by an attacker

The attacker can do anything Bob can do… might include everything from executing the BASH shell to mount a CD-ROM.

Under SELinux:

- the blinkled process would run in a narrowly defined domain of activity that would allow it to do its job;
- thus blinkled's privileges would not be determined based on its user/owner;
- rather, they would be determined by much more narrow criteria.

SELinux

94

With SELinux, the common Linux DACs still applies:

- if DAC blocks the action it's OK also for SELinux
- if DAC allows then SELinux evaluates it against its own security policies

The starting point of SELinux is the same as for DAC:

- "subjects" are processes (run user cmds)
- actions are "permissions"

But:

- subjects are just the processes that execute a command, not the users that run them
- objects not just files & dirs., but also user processes and system resources in kernel space and user space

SELinux

95

Classes of objects in SELinux:

- files
- dir
- socket
- tcp_socket
- unix_stream_socket
- file system
- node
- xserver
- cursor

Each class of objects has its own permissions (actions) over objects.

For example, for the dir class:

- search
- rmdir
- getattr
- remove_name
- reparent
- …

SELinux

96

It would be impossible to use if you had to create an individual rule for every possible subject against every possible object…

to manage complexity SELinux has:

1. "that which is not expressly permitted, is denied"
2. grouping of subjects, permissions, and objects

SELinux

97

By rule 1 you need to create rules/policies that describe the behaviors you expect and want, instead of all possible behaviors.

- by far, the most secure design principle any access control technology can have.
- but it needs to anticipate all possible allowable behavior and interactions …
- … by every daemon and command …

This is why some SELinux systems actually implements a policy:

"restrict only these particular services"

which frees all processes not explicitly covered in the policy.

- not the most secure way to use SELinux
- not the way SELinux was originally designed to be used.
- it's a justifiable compromise on general-purpose systems.

SELinux

98

each individual subject & object in SELinux is governed by a **security context** being a:

- **user** - individual user (human or daemon)
  SELinux maintains its own list of users
  user labels on subjects (processes) specify their privileges (associated to a SELinux user)
  user labels on objects specify their (SELinux user) owners
- **role** - like a group, assumed by users
  a user may only assume one role at a time,
  may only switch roles if and when authorized to do so
- **domain** - a "sandbox" being a combination of subjects and objects that may interact with each other

The model where each process (subject) is assigned to a domain, wherein only certain operations are permitted, is called **Type Enforcement** (TE) and it's the hearth of SELinux

Security Contexts

99

Suppose again we are securing the *blinkled* daemon with SELinux

- Recall it runs as the user *Bob*, reads the `file /home/someguy/messages.txt` and blinks out jokes written there in Morse code on the keyboard LED

we need an account "Bob" also in SELinux for *Bob* (in addition to the user "Bob" in Linux)
…and we need a role for Bob in this context:

- let it be `blink_r` ('_r' is a convention of SELinux)

Let's also call the domain of this security context `blinked_t` (again a SELinux convention). The domain specifies the rules:

- (*blinkled*) can read file `/home/someguy/messages.txt`
- (*blinkled*) can write file `/dev/numlocked`

The files `messages.txt` and `numlocked` also need their own security context

- both can use the `blinked_t` domain
- They also take the generic role "object_r"
  - they are objects and thus passive, hence they do not need a role, but in SELinux everything must have a role…

Security Contexts

100

two types of decisions:

access decisions

- when subjects do things to objects that already exist, or create new things in expected domain
- they are simple to deal with

transition decisions

- invocation of processes in different domains than the one in which the subject-process is running
- creation of objects in different types (domains) than their parent directories
- transitions must be authorized by SELinux policy, it's an important check against privilege-escalation attacks!

Decision Making in SELinux

101

Note:

A child process inherits the security context (user, role, domain) of the parent also in SELinux

At creation, a file inherits the security context of its parent directory

However, processes and files transitions are normal part of the system operation:

- a file may be needed in multiple domains
- a process may need to execute operations outside of its domain

SELinux ensures domains are not changed arbitrarily:

- Process transition in a new domain (or invocation of a process in another domain, …) allowed only if there's an explicit policy authorizing it
- Same for files: a process may change the security context of a file only if there's a policy that allows it to do so.

Decision Making in SELinux

102

Besides domain enforcement, SELinux also has **Role Based Access Control** (RBAC)

- rules specify **roles** a user may assume
- other rules specify circumstances when a user may **transition** from one role to another
- particularly useful when dealing with real users, not just processes

Comes relatively straightforward in SELinux, as the rules:

- specify what **roles** a human user can assume
- specify under what circumstances a user may change role (**transitions**)

Note: in SELinux RBAC a user can take one role at time. This is different than groups in the Linux DAC!

RBAC

103

SELinux also provides **Multi Level Security** (MLS)

concerns handling of classified data

"no read up, no write down"

MLS is enforced via file system labeling

Based on the Bell-LaPadula (BLP) model

MLS Controls

104

creating and maintaining SELinux policies is complicated and time-consuming

a single SELinux policy may consist of hundreds of lines of text

RHEL (a configuration tool for some distributions) has a default "targeted" policy

- defines types for selected network apps
- allows everything else to use DAC controls

SELinux offer a wide range of commands

- see additional references for details

SELinux Policy Management

105

Novell's Mandatory Access Control for SuSE Linux

- enforced at kernel level
- using Linux Security Modules

restricts behavior of selected applications in a very granular but targeted way

- hence a compromised root application's access will be contained
- has no controls addressing data classification
- hence only a partial MAC implementation

non-protected apps just use Linux DAC

Novell AppArmor

106

Summary

Review of the Linux OS

Review of Linux security model and DAC vulnerabilities

Linux hardening

MAC, SELinux

107