Linux/Unix Overview
and
Access Control

Victor Francisco Fonte
University of Minho
vff@di.uminho.pt

2015

Linux 101

- Multi-program, multi-task, multi-user

- Kernel, libraries, system calls, processes

- Super-user, regular user

- Shell, system and user-level command-line programs

- Configs and service management

- Optional GUI and graphical apps

Linux 101 (2)

- Program vs. process

- Virtual memory with address space isolation

- File system

- Inter-process communication

Processes

- Running program

- Isolated address space

- Executed under an user and group id

  - actual or virtual users

  - real or effective user and group ids

- May spawn other processes

Process: address space

- Traditional linear model

- Code (lower), data, and stack (higher)

- Data: initialised vs. non-initialised (e.g. heap)

- Access to the null (zero) address is illegal

Process: file descriptors

- File descriptor represents a kernel-level file-like object

- Processes inherited file descriptors: standard input, output, and error

- Processes inherit all opened file descriptors from their fathers

Process: creation and execution

- Processes are created by cloning

    - address space is copied (lazy)

    - opened file descriptors are inherited

- Programs are executed

    - address space is replaced by the contents of an executable file (ABI)

    - opened file descriptors are preserved

Inter-process Communication

- Files

- Traditional IPC:

  - shared memory

  - message queues

  - semaphores

- Pipes (named or un-named), sockets (local or network domains), …

Unix File and Directory Permissions and Modes

- POSIX-compliant systems

- POSIX 1:2008 a.k.a. Single Unix Specification (SUS) V4

Users, Groups and Passwords

- User ID (UID), group ID (GID)

- Primary and secondary groups

- User DB: /etc/passwd (and /etc/shadow)

- Group DB: /etc/group

Access Control to Resources

- Homogeneity, simplicity

- Files, directories, and other resources

- Ownership and permissions

Ownership

- Assigned user owner

- Assigned group owner

Classes of Users

1. User owner

2. Member of the owning group

3. Others (everyone else)

Checked by this order.

Permissions

- Read, write, execute

- For each class of user

- Permissions set independently

- Not all combinations make sense

- Files and directories are similar

- But slightly different semantics

Permissions

- umask: user-default restriction on permissions

- 12 bit permissions kept in resource i-node

- Permissions influence syscalls (not commands)

- Only super-user can modify ownership

- Only owner (or super-user) can modify perms

Permissions set on Files

- Read: access content

- Write: modify content

- Execute: execute content

Directories

- Directories are special files

- Map names to i-nodes

- Access to names protected by 'read'

- Access to i-nodes protected by 'execute'

Permissions set on Dirs

- Read: view/list names

- Write: add/delete/rename names

- Execute: chdir, access i-nodes (stat)

- Execute a.k.a search permission

- Execute permissions needed along a path

- Permissions not inherited from parent dir

Permissions and other Attributes

- perms, u-owner, g-owner, c/m-date, name, ...

- perms: type, perms, set uid/gid, sticky, ...

- ex: -rwxr--r--, drwxr-xr-x

- ex: drwsrwsrwt, drwSrwSrwT

- ex: -rwxrw-r--+

Permissions: Experimentation

- chmod, chown, chgrp

- read(), write(), exec(), unlink(), stat()

- strace

Permissions: Experimentation

- mkdir adir && touch adir/afile

- ls adir; ls -l adir; ls -l adir/afile

- chmod -x adir

- ls adir; ls -l adir; ls -l adir/afile

Real vs. Effective User & Group IDs

- Default: (EUID, EGID) = (RUID, RGID)

- If SUID, SGID are set on resources

  - EUID = resource user owner ID

  - EGID = resource group owner ID

SUID & SGID Attrs

- Useful but dangerous

- Violates the minimal privileges principle

- Lookout for root SUID

- In particular, root SUID + 'write' permission

- Executable/Not-executable: 's'/'S'

SUID & SGID on Files

- 'S' on user owner id: no meaning

- 'S' on group owner id: mandatory locking

- 's' has no effect on scripts... why?

SUID & SGID on DLLs

- Shared objects: .so, .so.*number*

- System-wide defaults: /etc...

- Override: LD_LIBRARY_PATH

- Dangerous for SUID, SGID... why?

- If SUID or SGID then LD_LIBRARY_PATH is ignored

Sticky Bit on Files

- Sticky a.k.a. Text Bit

- Code kept on swap or memory

- Deprecated in favor of virtual memory

- Executable/Non-executable: 't'/'T'

- No effect on non-executables

SUID & SGID on Dirs

- SUID no effect on directories

- On Linux & Solaris SGID: group owner copied to entries (no inheritance!)

- SGID semantics is not in SUS V4

Sticky Bit on Dirs

- If set, only owner of a resource, the owner of the directory (or the super-user) can move/rename/delete the resource

- Works in conjunction with 'write' perm

Access Control Lists

- Permission set to specific users and groups

- Complements the UGO/RWX mechanism

- Default ACL for directories can be inherited, but usually are just copied to entries

- POSIX proposal withdrawn

- Solution to the per directory umask problem

ACLs: Experimentation

- Utilities: setfacl, getfacl

- Effective vs real permission mask

- Denoted by a '+' after perms

Additional/Extended Attributes

- Additional attributes:

  - NTFS, ext-family, ...

  - ext2: lsattr, chattr

- Extended attributes:

  - name, value pairs

  - getfattr, setfattr

- A ' ' or '.' after perms unless ACL is set

Rootly Powers

- Some operations ignore permissions

- Check who makes the request: EUID = 0?

- Eg. system shutdown, port binding, ...

- Solutions: capabilities mechanism

- Program start as root but give up capabilities they don't need

- Minimize privileges to exploit

Chroot Jail

- changes the apparent root directory

- affects running process and its children

- operation restricted to the super-user

- usage: testing and development, dependency control and compatibility, recovery, privilege separation

Chroot best practices

- change working directory into the jail before chroot

- change real/effective user id to a non-root

- keep as little as possible inside the jail

- have root own as many jailed read-only files as possible

- limit all permissions of files and directories

- create a permissions-setting script

- chroot from inside the daemon itself (avoid wrapping)

Chroot best practices

- preload dynamically loaded objects

- avoid using the jailed /etc/passwd file

- close file descriptors aggressively before chrooting

- link config files from the outside

- update environment variables to reflect the new root

Minimal chroot

- close(*unused file descriptors*);

- chdir("jail");

- chroot(".");

- setuid(non-root-uid);

Chroot

- related system calls: chroot()

- related commands: chroot

- precursor of jail, Solaris Containers, Linux Containers, Docker, Linux userspaces (kernel 3.8+)

Further reading

- http://en.wikipedia.org/wiki/Unix_security

- http://www.tldp.org/LDP/intro-linux/html/sect_03_04.html

- http://www.cse.psu.edu/~trj1/cse497b-s07/slides/cse497b-lecture-18-unixsecurity.pdf