Linux Security Overview

Vítor Francisco Fonte, vff@di.uminho.pt, University of Minho, 2024

Overview

- Introduction to the Linux operating system
- Processes, users and groups
- Access control to file system resources
- Extended permissions and attributes
- Isolating part of the filesystem
- Other bits related to the filesystem

The Linux Operating System

- Free and open-source, general-purpose operating system
  - Multiple architectures, distributions
- Wide range of features and applications
  - Broad attack surface (vulnerabilities are regularly discovered)
- Discretionary Access Control (DAC) model
  - Significant shortcomings
- Secure deployment and operation is possible leveraging:
  - Native security controls
  - Additional security features
  - Best practices for mitigating threats

The Monitor Reference Model

- Subject, Object, Action and Access Control Metadata
- TODO: diagram

Access Control to the Linux File System

- Subject:
  - An actual human user or a virtual user
  - A process executes on behalf of a subject
- Object:
  - A file or a directory
  - Belongs to a user and a group
  - Ownership can only be changed by the “root” user
- Action:
  - Read, write, and execute
  - Different semantics for files and directories

Access Control to the Linux File System

- Access Control Metadata:
  - An access control list (ACL) for each object set by the corresponding owner
  - Permission to perform each available action (R, W, X) for the following access cases:
    1. The user owning the object (U)
    2. A user belonging the group that owns the object (G)
    3. All other users (O)
- Authorisation and execution of the action:
  - The reference monitor first classifies the subject according to ordered list above
  - Authorisation is granted if permission for the action is set for access case being assessed

Linux 101

- Multi-programming, -tasking, -threading, -processing, -architecture and -user
- Large, configurable monolithic kernel (makefile, module support, 30+ million SLoCs)
- Virtual memory with address space isolation (by default)
- Follows POSIX (with caveats): APIs, service infrastructure, access control, etc.
- Efficient, fair and secure use of (real and virtual) resources
- Kernel space vs. user space
- Kernel vs. system calls vs. processes
- Multiple extensions and specialisations: domain-specific, security-specific, etc.

Linux 101

- Program vs. process
- Super-user vs. regular user
- Real vs. virtual user Kernel config and service management
- Optional GUI and graphical applications
- User processes:
  - shell (interpreter), system and user-level command-line programs
- File system and inter-process communication

Processes

- Created by cloning an existing process
  - may spawn other processes
- Isolated, linear address space
  - lazy copy of the parent's address space
  - code (lower), data (middle), and stack (higher)
  - data: initialised vs. non-initialised (e.g. heap)
  - access to the null (zero) address is illegal
- Executed under a user and group id
  - actual or virtual users
  - real or effective user and group ids
- Maintains a set of opened file descriptors
  - a file descriptor represents a kernel-level file-like object
  - can be inherited from the parent process (including stdin, out, err)
  - can be opened, duplicated, read, write, closed, etc.
- Can execute other programs in the filesystem
  - 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), …
- Uniform access control mechanisms (more or less)

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

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

- Homogeneity, simplicity
  - files, directories, and other resources
- Ownership and permissions
  - assigned user owner
  - assigned group owner
- Classes of users (checked in this exact order):
  1. User owner
  2. Member of the owning group
  3. Others (everyone else)

Permissions

- Read, write, execute
- For each class of user
- Set independently
- Not all combinations make sense
- Files and directories are similar
- But slightly different semantics

- 12 bit kept in resource i-node
- Permissions influence syscalls (not commands)
- Only super-user can modify ownership
- Only owner (or super-user) can modify perms
- User mask: user-default restriction on permissions (umask)

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 Files and Directories

- Files:
  - Read: access content
  - Write: modify content
  - Execute: execute content
- Directories:
  - Read: view/list names
  - Write: add/delete/rename names
  - Execute: chdir, access i-nodes (stat)
- Observations:
  - Execute a.k.a search permission
  - Execute permissions needed along a path
  - Permissions not inherited from parent dir

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 Attributes

- 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, DLLs and Directories

- Files:
  - ’S’ on user owner id: no meaning
  - ‘S’ on group owner id: mandatory locking
  - ‘s’ has no effect on scripts... why?
- Dynamic Linking Libraries (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
- Directories:
  - 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 Files and Directories

- 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
- Directories:
  - 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 (ACLs)

- 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
- Experimentation:
  - Utilities: setfacl, getfacl
  - Effective vs real permission mask
  - Denoted by a ‘+’ after perms

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

Change Root

- 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
- Inspiration for later developments on contentorization: precursor of jail, Solaris Containers, Linux Containers, Docker, Linux userspaces (kernel 3.8+)

Change Root: 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)
- Preload dynamically loaded objects
- Avoid using the jailed /etc/passwd file
- Close file descriptors aggressively before chrooting
- Lnk config files from the outside
- Update environment variables to reflect the new root

Minimal Chroot Example

- close(unused file descriptors);
- chdir("jail");
- chroot(".");
- setuid(non-root-uid);

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