

# Linux security mechanisms



## Mechanisms

- ▷ Capabilities
- ▷ cgroups (control groups)
- ▷ LSM (Linux Security Modules)



# Linux management privileges

## ▷ Initial UNIX philosophy

- ♦ Privileged processes (UID = 0)
  - Bypass all kernel permission checks
- ♦ Unprivileged processes (UID ≠ 0)
  - Subject to permission checking based on their credentials
  - Effective UID, effective GID, secondary group list



# Capabilities

- ▷ Protection mechanism introduced in Kernel 2.2
- ▷ They allow to divide the traditional super-user privileges into distinct units
  - ♦ That can be independently enabled and disabled
- ▷ Capabilities are a per-thread attribute
  - ♦ Propagated through forks
  - ♦ Changed explicitly of by execs



## List of capabilities: Examples (small sample ...)

- ▷ CAP\_CHOWN
  - ♦ Make arbitrary changes to file UIDs and GIDs
- ▷ CAP\_DAC\_OVERRIDE / CAP\_DAC\_READ\_SEARCH
  - ♦ Bypass file permission / directory transversal checks
- ▷ CAP\_KILL
  - ♦ Bypass permission checks for sending
- ▷ CAP\_NET\_ADMIN
  - ♦ Perform various network-related operations
- ▷ CAP\_SYS\_ADMIN
  - ♦ Overloaded general-purpose administration capability



## Capability management

- ▷ Per-thread capabilities
  - ♦ They define the privileges of the thread
  - ♦ Divided in sets
- ▷ Sets
  - ♦ Effective
  - ♦ Inheritable
  - ♦ Permitted
  - ♦ Bounding
  - ♦ Ambient



## Thread capability sets: Effective

- ▷ Set of capabilities used by the kernel to perform permission checks for the thread



## Thread capability sets: Inheritable

- ▷ Set of capabilities preserved across an exec
  - ◆ Remain inheritable for any program
- ▷ Are added to the permitted set when executing a program that has the corresponding bits set in the file inheritable set



## Thread capability sets: Permitted

- ▷ Limiting superset
  - ♦ For the effective capabilities that the thread may assume
  - ♦ For the capabilities that may be added to the inheritable set
    - Except for threads w/ CAP\_SETPCAP in their effective set
- ▷ Once dropped, it can never be reacquired
  - ♦ Except upon executing a file with special capabilities



## Thread capability sets: Bounding

- ▷ Set used to limit the capabilities that are gained during an exec
  - ♦ From a file with capabilities set
- ▷ Was previously a system-wide attribute
  - ♦ Now is a per-thread attribute



## Thread capability sets: Ambient

- ▷ Set of capabilities that are preserved across an exec of an unprivileged program
  - ♦ No set-UID or set-GID
  - ♦ No capabilities set
- ▷ Executing a privileged program will clear the ambient set



## Thread capability sets: Ambient

- ▷ No capability can ever be ambient if it is not both permitted and inheritable
  - ♦ One cannot preserve something one does not have
  - ♦ One cannot preserve something one cannot inherit
  - ♦ Automatically lowered if either of the corresponding permitted or inheritable capabilities is lowered
- ▷ Ambient capabilities are added to the permitted set and assigned to the effective set upon a exec



## Files extended attributes (xattr)

- ▷ Files' metadata in UNIX-like systems
  - ♦ Some not interpreted by kernels
- ▷ Linux: key-value pairs
  - ♦ Keys can be defined or undefined
  - ♦ If defined, their value can be empty or not
  - ♦ Key's namespaces
    - namespace.attr\_name[attr\_name]
- ▷ Namespace classes
  - ♦ Security
    - For files' capabilities
    - `setcap` / `getcap`
  - ♦ System
    - ACL
  - ♦ Trusted
    - Protected metadata
  - ♦ User
    - `setfattr` / `lsattr` / `getfattr`



## File capabilities

- ▷ Stored in the `security.capability` attribute
- ▷ Specify capabilities for threads that exec a file
  - ♦ Permitted set
    - Immediately forced into the permitted set
    - Previous AND with the thread's bounding set
  - ♦ Inheritable set
    - To AND with the threads inheritable set
    - Can be used to reduce the effective set upon the exec
  - ♦ Effective bit
    - Enforce all new capabilities into the thread's effective set



## Capability transfer across exec: No privileged files

- ▷ P'(ambient)  
P(ambient)
- ▷ P'(permitted)  
P'(ambient)
- ▷ P'(effective)  
P'(ambient)
- ▷ P'(inheritable)  
P(inheritable)
- ▷ P'(bounding)  
P(bounding)



## Capability transfer across exec (non-root)

- ▷ P'(ambient)  
(file is privileged) ? 0 : P(ambient)
- ▷ P'(permitted)  
(P(inheritable) & F(inheritable)) |  
(P(bounding) & F(permitted)) |  
P'(ambient)
- ▷ P'(effective)  
F(effective) ? P'(permitted) : P'(ambient)
- ▷ P'(inheritable)  
P(inheritable)
- ▷ P'(bounding)  
P(bounding)



## Capability transfer across exec (root)

- ▷ EUID = 0 or RUID = 0
  - ♦ File sets are considered to be all 1's
- ▷ EUID = 0
  - ♦ File effective bit considered 1
- ▷ Exception: EUID = 0, RUID  $\neq$  0
  - ♦ File capabilities are honored if present



## Control groups (cgroups)

- ▷ Collection of processes bound by the same criteria and associated with a set of parameters or limits
- ▷ cgroups are organized hierarchically
  - ♦ cgroup file system
  - ♦ Limits can be defined at each hierarchical level
    - Affecting the subhierarchy underneath
- ▷ Subsystems
  - ♦ Kernel component that modifies the behavior of cgroup processes
  - ♦ Resource controllers (or simply controllers)



## cgroups file system

- ▷ This file system is created by mounting several controllers as cgroup-type file system entities
  - ♦ Usually `/sys/fs/cgroup`
- ▷ Each controller defines a tree of cgroups below the mount point
  - ♦ e.g. memory controller → `/sys/fs/cgroup/memory`



## cgroups v1 and v2

- ▷ Currently two versions coexist
  - ♦ But controllers can only be used in one of them



## cgroup controllers

- ▷ **cpu, cpucct**
  - ♦ CPU usage & accounting
- ▷ **cpuset**
  - ♦ CPU bounding
- ▷ **memory**
  - ♦ Memory usage & accounting
- ▷ **devices**
  - ♦ Device creation & usage
- ▷ **freezer**
  - ♦ Suspend/resume groups of processes
- ▷ **net\_cls**
  - ♦ Outbound packet classification
- ▷ **blkio**
  - ♦ Block I/O management
- ▷ **perf\_event**
  - ♦ Performance monitoring
- ▷ **net\_prio**
  - ♦ Network interfaces priorities
- ▷ **hugelb**
  - ♦ Huge pages management
- ▷ **pids**
  - ♦ # of processes in cgroup
- ▷ **rdma**
  - ♦ RDMA / IB resources' management



## cgroups v1: Common files

- ▷ **cgroup.procs**
  - ♦ The processes in the cgroup



## cgroups of a process

- ▷ A process can be controlled by an arbitrary number of cgroups
- ▷ The list of a process' cgroups is given by the `/proc` file system
  - ♦ `/proc/[PID]/cgroup`



## Linux Security Modules (LSM)

- ▷ Framework to add new Mandatory Access Control (MAC) extensions to the kernel
- ▷ Those extensions are not kernel modules
  - ♦ They are embedded in the kernel code
  - ♦ They can be activated or not at boot time
  - ♦ List of extensions given by `/sys/kernel/security/lsm`



## LSM extensions

- ▷ Capacities (default)
- ▷ AppArmor
- ▷ LoadPin
- ▷ SELinux
- ▷ Smack
- ▷ TOMOYO
- ▷ Yama



## AppArmor

- ▷ Enables the definition of per-application MAC policies
  - ♦ Profiles
  - ♦ Applications are identified by their path
    - Instead of i-node
- ▷ Profiles restrict applications' actions to the required set
  - ♦ All other actions will be denied
- ▷ Profiles define
  - ♦ Capabilities
  - ♦ Permissions (r/w/x) over file system objects



## AppArmor: profiles

- ▷ Profiles are loaded into the kernel
  - ♦ Upon compilation from textual files
  - ♦ `apparmor_parser`



## AppArmor: Usage of loaded profiles

- ▷ Enforce mode
  - ♦ Auditing rules are enforced
  - ♦ Status events (profile loads, ...) and denied events generate audit messages
- ▷ Complain mode
  - ♦ Auditing rules are checked
    - But all denied events succeed
  - ♦ Denied events generate allowed audit messages



## AppArmor: Usage of loaded profiles

### ▷ Audit mode

- ♦ Generates a message for each event mediated by AppArmor
  - Whether it is allowed or rejected

### ▷ Kill mode

- ♦ Generates a message for any denied event and then kills the process

