Linux
security mechanisms

Mechanisms

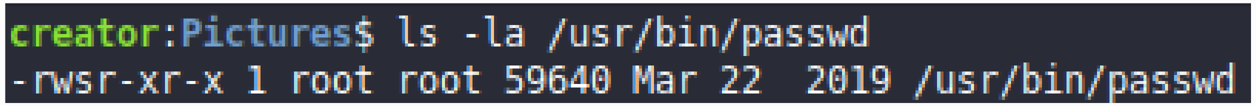
- Capabilities
- cgroups (control groups)
- LSM (Linux Security Modules)

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

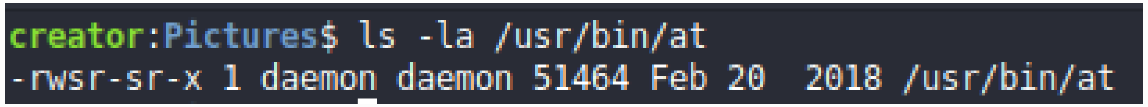
Unix file protection ACLs:
Special protection bits

- Set-UID bit
  - Is used to change the UID of processes executing the file



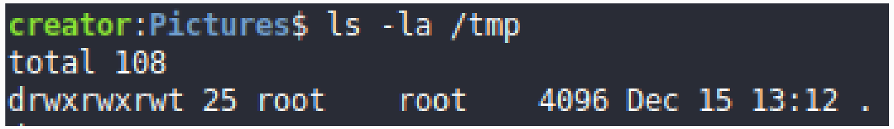
```
creator:Pictures$ ls -la /usr/bin/passwd
-rwsr-xr-x 1 root root 59640 Mar 22  2019 /usr/bin/passwd
```

- Set-GID bit
  - Is used to change the UID of processes executing the file



```
creator:Pictures$ ls -la /usr/bin/at
-rwsr-sr-x 1 daemon daemon 51464 Feb 20  2018 /usr/bin/at
```

- Sticky bit
  - Hint to keep the file/directory as much as possible in memory cache



```
creator:Pictures$ ls -la /tmp
total 108
drwxrwxrwt 25 root    root    4096 Dec 15 13:12 .
```

Privilege elevation:
Set-UID mechanism

- Change the effective UID of a process running a program stored on a Set-UID file
  - If a program file is owned by UID X and the set-UID bit of its ACL is set, then it will be executed in a process with UID X
    - Independently of the UID of the subject that executed the program

- Allows normal users to execute privileged tasks encapsulated in administration programs
  - Change the user's password (passwd)
  - Change to super-user mode (su, sudo)
  - Mount devices (mount)

Privilege elevation:
Set-UID mechanism (cont.)

- Effective UID / Real UID
  - Real UID (rUID) is the UID of the process creator
    - App launcher
  - Effective UID (eUID) is the UID of the process
    - The one that really matters for defining the rights of the process
    - eUID may differ from rUID

- UID change
  - Ordinary application
    - eUID = rUID = UID of process that executed **exec**
    - eUID cannot be changed (unless = 0)
  - Set-UID application
    - eUID = UID of **exec**'d application file, rUID = initial process UID
    - eUID can revert to rUID
  - rUID cannot change

Privilege elevation:
Set-UID/Set-GID decision flowchart

- exec (path, …)
  - File referred by path has Set-UID?
  - Yes
    - ID = path owner
    - Change the process effective UID to ID of path owner
  - No
    - Do nothing

  - File referred by path has Set-GID?
  - Yes
    - ID = path GID
    - Change the process GIDs to ID only
  - No
    - Do nothing

Capabilities

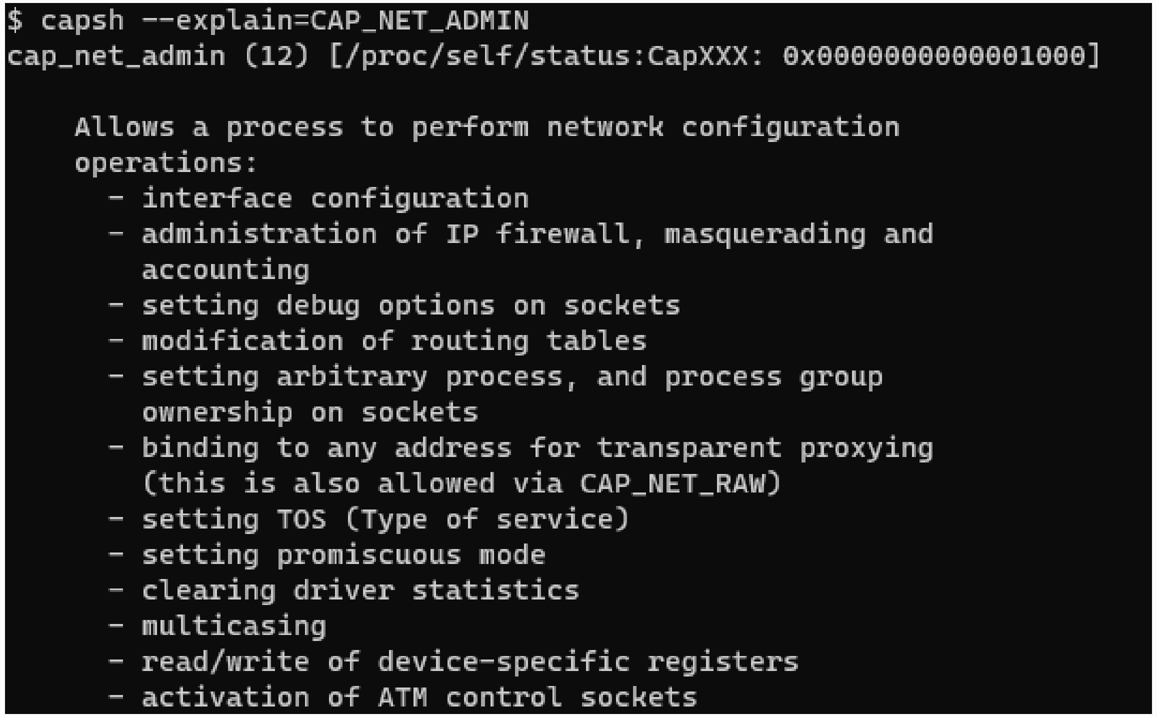
- Protection mechanism introduced in Kernel 2.2

- 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 signals
- CAP_NET_ADMIN
  - Perform various network-related operations
- CAP_SYS_ADMIN
  - Overloaded general-purpose administration capability



```
$ capsh --explain=CAP_NET_ADMIN
cap_net_admin (12) [/proc/self/status:CapXXX: 0x0000000000001000]

    Allows a process to perform network configuration
    operations:
      - interface configuration
      - administration of IP firewall, masquerading and
        accounting
      - setting debug options on sockets
      - modification of routing tables
      - setting arbitrary process, and process group
        ownership on sockets
      - binding to any address for transparent proxying
        (this is also allowed via CAP_NET_RAW)
      - setting TOS (Type of service)
      - setting promiscuous mode
      - clearing driver statistics
      - multicasing
      - read/write of device-specific registers
      - activation of ATM control sockets
```

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

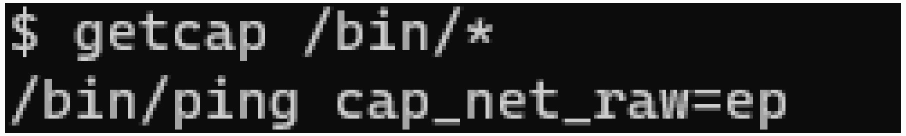
- That is: these are the effective capabilities being used

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



```
$ getcap /bin/*
/bin/ping cap_net_raw=ep
```

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

- Ambient capabilities must be both permitted and inheritable
  - One cannot preserve something one cannot 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 an exec

Files extended attributes (xattr)

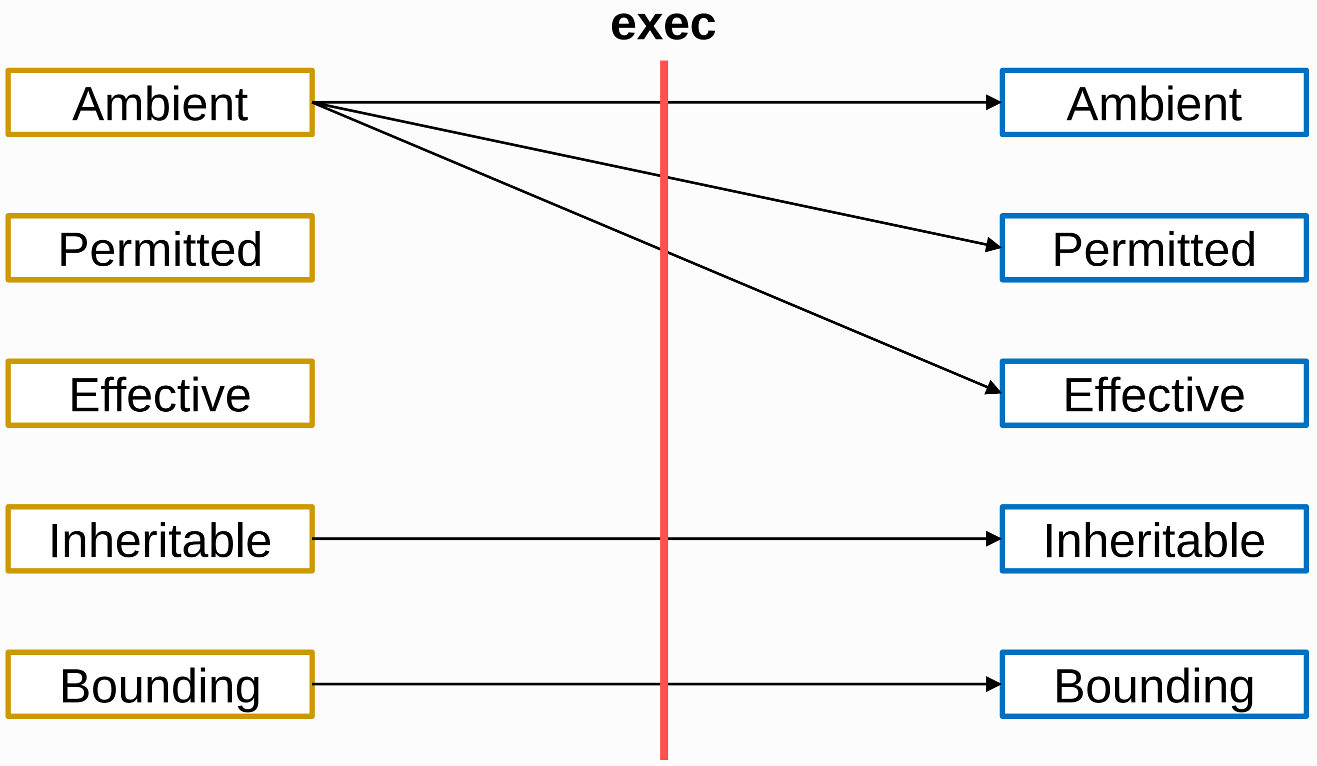
- Files' metadata in UNIX-like systems
  - Some not interpreted by kernels
- Linux: key-value pairs
  - Keys can be defined or undefined
  - If defined, their value can be empty or not
  - Key's namespaces
    - `namespace.attr_name[.attr_name]`
- Namespaces
  - security
    - For files' capabilities
    - setcap / getcap
  - system
    - ACL
  - trusted
    - Protected metadata
  - user
    - setfattr / lsattr / getfattr

File capabilities

- Stored in the security.capability attribute

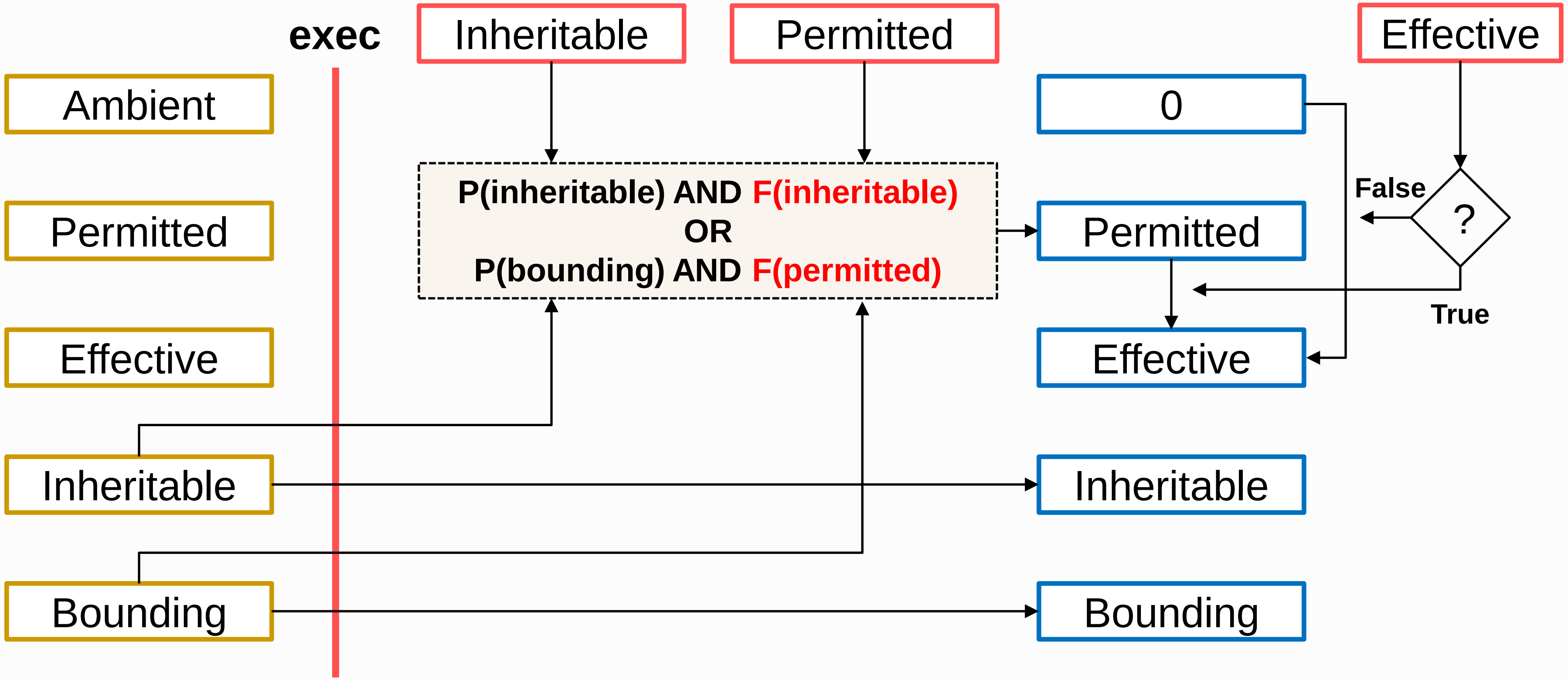
- Specify capabilities for threads that exec a file
  - Permitted set
    - Immediately forced into the permitted set
    - Previous AND with the thread’s bounding set
  - Inheritable set
    - To AND with the threads’ inheritable set
    - Can be used to reduce the effective set upon the exec
  - Effective bit
    - Enforce all new capabilities into the thread’s effective set

Capability transfer across exec: No privileged files



Capability transfer across exec (non-root)

Privileged files



Capability transfer across exec (root)

- EUID = 0 or RUID = 0
  - Capability sets are considered to be all 1’s

- EUID = 0
  - File effective bit considered 1

- Exception: EUID = 0, RUID ≠ 0
  - Set-UID file was executed
  - 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 sub-hierarchy underneath

- Subsystems
  - Kernel component that modifies the behavior of cgroup processes
  - Resource controllers (or simply controllers)

cgroups v1 and v2

- Currently two versions coexist
  - But controllers can only be used in on of them

cgroups file system

- This file system is created by mounting several controllers as cgroup-type file system entities
  - Usually /sys/fs/cgroup
  - In V2 all controllers are part of a single cgroup2

- Each controller defines a tree of cgroups below the mount point
  - e.g. memory controller → /sys/fs/cgroup/…/memory.[…]

cgroup V2 (and V1) controllers

- cpu (cpu & cpuacct in V1)
  - CPU usage & accounting
- cpuset
  - CPU bounding
- memory
  - Memory usage & accounting
- devices
  - Device creation & usage
- freezer
  - Suspend/resume groups of processes
- Io (blkio in V1)
  - Block I/O management
- perf_event
  - Performance monitoring
- hugelb
  - Huge pages management
- pids
  - # of processes in cgroup
- rdma
  - RDMA / IB resources’ management

- Deprecated from V1
  - net_cls
    - Outbound packet classification
  - net_prio
    - Network interfaces priorities

cgroup V2 definition

- Directory under /sys/fs/cgroup
  - With a set of controllers defined by cgroup.controllers
  - With hierarchy limits defined by cgroup.depth and cgroup.descendants
  - With files to send KILL signals (cgroup.kill) and freeze/unfreeze orders (cgroup.freeze) to all cgroup processes
    - Including descendants
  - The processes using the cgroup are given by cgroup.procs and their status reported by cgroups.events
    - We can add a process to a cgroup just by writing its PID on the first file

- For each active controller, specific files will exist

- Processes can only belong to leaf cgroups
  - “No internal processes” rule

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

- Capabilities (default)
- AppArmor
  - MAC for applications
- LoadPin
  - Kernel-loaded files origin enforcement
- SELinux
- Smack
  - Simplified Mandatory Access Control Kernel
- TOMOYO
  - Name-based MAC extension
- Yama
  - System-wide DAC security protections that are not handled by the core kernel itself
- SafeSetID
  - Restricts UID/GID transitions

Source: https://www.kernel.org/doc/html/next/admin-guide/LSM/index.html

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
  - Actions white-listed
  - Logging actions

AppArmor: profiles

- Profiles are loaded into the kernel
  - Upon compilation from textual files
  - apparmor_parser
- Profiles can be used on a voluntary basis
  - aa-exec

Confinement: Namespaces

- Allows partitioning of resources in views (namespaces)
  - Processes in a namespace have a restricted view of the system
  - Activated through syscalls by a simple process:
    - clone: Defines a namespace to migrate the process to
    - unshare: disassociates the process from its current context
    - setns: puts the process in a Namespace
- Types of Namespaces
  - **Mount**: Applied to mount points
  - **process id**: first process has id 1
  - **network**: "independent" network stack (routes, interfaces…)
  - **IPC**: methods of communication between processes
  - **uts**: name independence (DNS)
  - **user id**: segregation of permissions
  - **cgroup**: limitation of resources used (memory, cpu…)

```
## Create netns named mynetns
root@vm: ~# ip netns add mynetns

## Change iptables INPUT policy for the netns
root@linux: ~# ip netns exec mynetns iptables -P INPUT DROP


## List iptables rules outside the namespace
root@linux: ~# iptables -L INPUT
Chain INPUT (policy ACCEPT)
target     prot opt source               destination


## List iptables rules inside the namespace
root@linux: ~# ip netns exec mynetns iptables -L INPUT
Chain INPUT (policy DROP)
target     prot opt source               destination
```

```
## List Interfaces in the namespace
root@linux: ~# ip netns exec mynetns ip link list
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 100
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00


## Move the interface enp0s3 to the namespace
root@linux: ~# ip link set enp0s3 netns mynetns

## List interfaces in the namespace
root@linux: ~# ip netns exec mynetns ip link list
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 100
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT...
    link/ether 08:00:27:83:0a:55 brd ff:ff:ff:ff:ff:ff


## List interfaces outside the namespace
root@linux: ~# ip link list
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT...
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

Confinement: Containers

- Explores namespaces to provide a virtual view of the system
  - Network isolation, cgroups, user ids, mounts, etc…
- Processes are executed under a container
  - Container is an applicational construction and not of the core
  - Consists of an environment by composition of namespaces
  - Requires building bridges with the real system network interfaces, proxy processes
- Relevant approaches
  - **LinuX Containers**: focus on a complete virtualized environment
    - evolution of OpenVZ
  - **Docker**: focus on running isolated applications based on a portable packet between systems
    - uses LXC