CONFIGURATION RECOMMENDATIONS OF A GNU/LINUX SYSTEM

ANSSI GUIDELINES

ANSSI-BP-028-EN
22/02/2019

TARGETED AUDIENCE:

Developers **Administrators** **IT security managers** IT managers Users

Information

Warning

This document, written by ANSSI, the French National Information Security Agency, presents the **"Configuration recommendations of a GNU/LINUX system"**. It is freely available at www.ssi.gouv.fr/en/.
It is an original creation from ANSSI and it is placed under the "Open Licence v2.0" published by the Etalab mission [12].
According to the Open Licence v2.0, this guide can be freely reused, subject to mentionning its paternity (source and date of last update). Reuse means the right to communicate, distribute, redistribute, publish, transmit, reproduce, copy, adapt, modify, extract, transform and use, including for commercial purposes
These recommendations are provided as is and are related to threats known at the publication time. Considering the information systems diversity, ANSSI cannot guarantee direct application of these recommendations on targeted information systems. Applying the following recommendations shall be, at first, validated by IT administrators and/or IT security managers.
This document is a courtesy translation of the initial French document **"Recommandations de configuration d'un système GNU/Linux"**, available at www.ssi.gouv.fr. In case of conflicts between these two documents, the latter is considered as the only reference.

Document changelog:

VERSION	DATE	CHANGELOG
1.2	22/02/2019	English version based on a courtesy translation provided by Red Hat

Contents

1
Glossary

- **AIDE** *Advanced Intrusion Detection Environment*
- **API** *Application Programming Interface*
- **ASLR** *Address Space Layout Randomisation*
- **AVC** *Access Vector Cache*
- **BIOS** *Basic Input/Output System*
- **CERT** *Computer Emergency Response Team*
- **DAC** *Discretionary Access Control*
- **GID** *Group IDentifier*
- **HIDS** *Host Intrusion Detection System*
- **HSM** *Hardware Security Module*
- **IOMMU** *Input/Output Memory Management Unit*
- **IPC** *Inter-Process Communication*
- **LDAP** *Lightweight Directory Access Protocol*
- **LSM** *Linux Security Module*
- **MAC** *Mandatory Access Control*
- **NSS** *Name Service Switch*
- **NX** *No-eXecute*
- **OS** *Operating System*
- **PAM** *Pluggable Authentication Module*
- **PIC** *Position Independent Code*
- **PID** *Process IDentifier*
- **PIE** *Position Independent Executable*
- **RBAC** *Role Based Access Control*
- **RSS** *Really Simple Syndication*
- **UID** *User IDentifier*
- **XD** *eXecute Disable*

2
Preamble

2.1 Foreword

Today Unix operating systems and derivatives, including GNU/Linux, are playing an important role in the ecosystem of equipments, systems, networks and telecommunications. They are widely deployed in several equipments (switches, routers, TV systems, vehicules…).

Their diversity and their composition are such that they are used according to a large number of possible combinations. Adressing in detail each possible use case is not the point of this guide. However some configuration rules allow to get a reasonably safe system if certain fundamental principles are respected, and to methodologically verify that they are properly applied, by using (for example) a checklist.

This guide focuses mainly on generic system configuration guidelines and on common sense principles that need to be applied during the deployment of hosted services. The configuration of these services themselves can be subject to a dedicated technical note, for example the *Recommendations for websites security hardening* [2] or *(Open)SSH secure use recommendations* [3] on the ANSSI website. The reader is invited to refer to the document *Recommendations for setting up system partitioning* [8] for a more specific and detailed approach of this subject.

It is advisable to study the applicability and the maintainability of each recommendation to the considered use case. It is also strongly recommended to have recourse to the skills of an expert in GNU/Linux systems for the implementation of these good practices.

2.2 Hardening Level

GNU/Linux distributions are highly heterogeneou, the control of the system platform is a complex task; expertise becomes really necessary as the number of services and servers increases. However, some hardening measures can be implemented based on the expected security level, which will depend on the sensitivity of the data handled or hosted by the system and the robustness of access controls realized in order to access to resources.

A publicly exposed service with low access control and handling sensitive data (transfer email server, corporate web server, etc.) requires an enhanced security level, perhaps even high. Conversely, a backup server on an isolated network and accessible only to a few people may require a lower level of security.

The recommendations of this guide are given based on an estimated level of hardening. This level does not necessarily depend on the difficulty and time required to deploy a given recommendation,

elements that can only be appreciated after an audit in a given context. These levels should be seen as guiding principle to help with the reading and with the system administration.

Niveau	Description
M I E H	**Minimal** level recommendations. To be implemented systematically on every system.
I E H	Recommendation applicable from the **intermediary** level. Corresponds generally to services protected by several layers of higher-level security.
E H	Recommendation applicable from the **enhanced** level. Generally for systems exposed to non-authenticated flows or from many sources.
H	Recommendation applicable to the **high** level. Corresponds to systems hosting sensitive data accessible from non-authenticated or poorly controlled networks.

Table 2.1: Hardening Level Reading Table

Depending on the retained hardening level, the recommendations apply from the minimum to the maximum target level. For example :

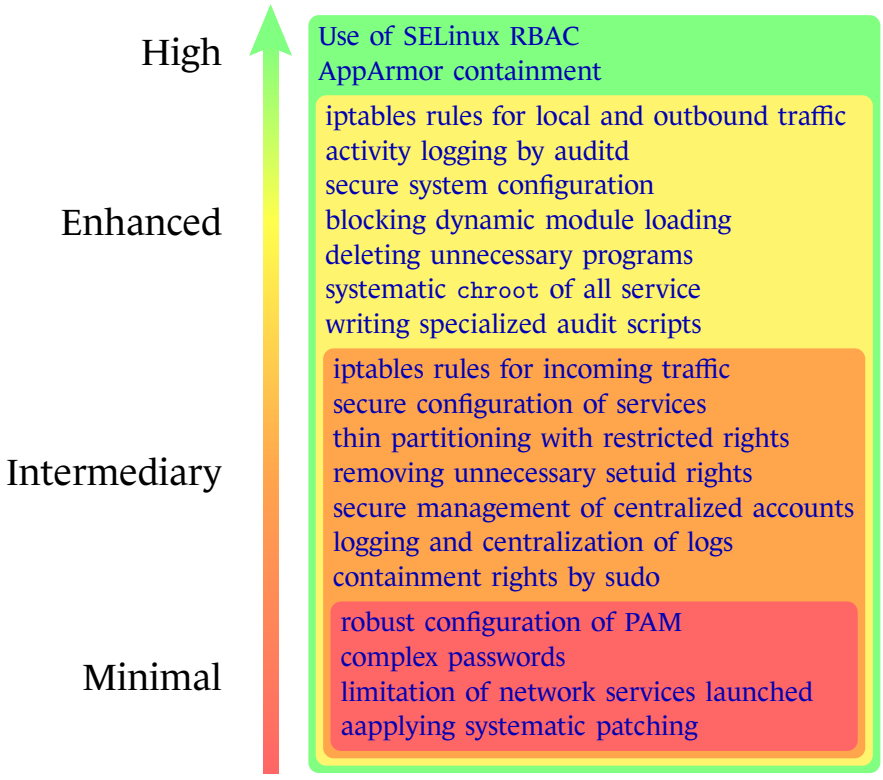


Figure 2.1: System Security Levels

3
General principles of security and hardening

Objective

Guide the installation and configuration of an operating system through different essential principles.

3.1 Principle of minimization

This principle indicates that deployed systems shoud avoid as much as possible unnecessary complexity, in order to:

- Reduce the attack surface to a minimum;
- Allow updating and effective system monitoring;
- Permit a more accessible monitoring activity of systems, the number of components to be monitored being reduced.

The implementation of this principle is sometimes difficult because it can quickly become in contradiction with others, equally important. Only a case study with the help of a system and security expertise will allow to make reasonable choices. The following chapters give targeted recommendations following the considered parts of the system.

R1

Minimization of installed services

Only the components strictly necessary to the service provided by the system should be installed.

Any service (especially in active listening on the network) is a sensitive element. Only those known and required for the operation and the maintenance must be installed. Those whose presence can not be justified should be disabled, removed or deleted.

A common case is the presence of resident autoconfiguration services, like DHCP or Zeroconf. Besides the fact that they can disrupt the network on which they are connected, these services can receive illegitimate data. Wherever possible and except operational need, they do not have to run on a server.

Services are often installed with default configurations that enable features potentially problematic from a security point of view. For example, SSH port forwarding that are often accepted and then

used to bypass firewall rules, or an Apache web server with directory indexing enabled and allowing to navigate in the directory tree system.

R2

Minimization of configuration

The features configured at the level of launched services should be limited to the strict minimum.

3.2 Principle of least privilege

This principle defines that any object or entity managed by a system has only rights strictly necessary for its execution, and nothing more.

The goal is both a gain in safety and security:

- the consequences of dysfunctions or vulnerabilities are limited to the privileges granted;
- the alteration and / or compromise of the system requires a privileges escalation, less trivial and discreet to achieve in cases where multiple layers of protection are set up.

R3

Principle of least privilege

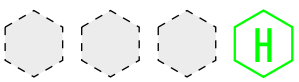
The services and executables available on the system must be analyzed in order to know the privileges they require, and must then be configured and integrated to use the bare necessities.

Unix and GNU/Linux distributions have a privilege separation that is mainly based on the two-level user concept: « classic » users and administrators (commonly called `root`).

If this separation lacks granularity, especially today where a user has a broad enough access to system primitives (while being more and more exposed to attacks: theft of accounts, leakage of information, etc.), the in-depth analysis of an expert is often required to highlight residual vulnerabilities with the aim of minimizing as much as possible the consequences.

Other mechanisms have emerged to allow additional, more granular rights control (SELinux, AppArmor…), but the user-based privilege separation (and its groups) is still the most frequently used. It can be accompanied by partitioning mechanisms (container, `chroot`, etc.) or filtering (*seccomp*) to limit access to the underlying operating system.

R4



Using access control features

It is recommended to use the mandatory access control (MAC) feature in addition to the traditional Unix user model (DAC), or possibly combine them with partitioning mechanisms.

Partitioning mechanisms are numerous, and each offers advantages and disadvantages. Some like `chroot` are universal and will work on all known Unix systems, but are limited to partitioning only part of the system (for `chroot`: access to the filesystem).

More advanced partitioning and privilege control functions than `chroot` exist under Linux; examples are capabilities (*POSIX capabilities*), namespaces, *seccomp* filters, or container-based solutions (LXC, Docker, VServer…).

3.3 Defense in depth principle

The principle of defense in depth relies on the design of several independent and complementary layers of security to delay an attacker whose mission is to compromise the system.

Each layer of security is therefore a point of resistance that the attacker must cross. Crossing a layer is accompanied by signals, alarms or log messages that allow to detect a suspicious activity and to be able to react to it. The remediation stage is also facilitated by additional aggregated information on the context of the compromise.

This principle has a real advantage: detection, ease of remediation, and improvement of security.

R5

Defense in depth principle

Under Unix and derivatives, defense in depth must be based on a combination of barriers that must be kept independent of each other. For example :

- authentication is necessary before carrying out operations, especially when they are privileged;
- centralized logging of events at the systems and services level;
- preferential use of services that implement partitioning mechanisms and / or separation of privileges;
- use of exploits prevention mechanisms.

R6

Network services partitioning

Network services should as much as possible be hosted on isolated environments. This avoids having other potentially affected services if one of them gets compromised under the same environment.

The terminology « environnement » is to be taken broadly here: it represents all resources and data accessible to the service depending on its privileges. The partitioning of environments can be achieved in different ways (dedicated user account, dedicated container, physical or virtual machine, etc.), each way has Pros and Cons that need to be studied: a dedicated user account can always access resources on which rights are incorrectly set (an audit of access rights must therefore be done), whereas containers and virtual machines offer a more efficient partitioning but can require additional integration efforts.

3.4 Monitoring and maintenance activity

Every system must be monitored to control drifts. The same goes for its maintenance in secure conditions.

R7

Logging of service activity

The activities of the running system and services must be logged and archived on an external, non-local system.

Software patches can bring new features to the software environment, thus contributing to raising its level of security. Others aim to rectify vulnerabilities present on the system. A monitoring activity on the corrective measures to be applied is essential, because it allows to know the vulnerabilities to which it is or has been exposed, and thus to lend special attention until a patch is available.

R8

Regular updates

It is recommended to have a regular and responsive security update procedure.

The monitoring activity can be done by the inscription to mailing lists (security teams of installed components and applications and their editors, RSS feeds of CERTs [a], and so on).

a. CERT-FR [10] website, among others.

4
Hardware configuration before installation

Objective

Enable or disable some hardware options in order to harden the platform that will be used for installation. This setting should preferably be done before installation so the system can take it into account as soon as possible.

The following recommendations apply to x86 architectures. The approach remains however applicable to other architectures except that the configuration mechanisms and directives will certainly be different.

4.1 General BIOS setting

The BIOS (and its modern equivalent: UEFI) is the main hardware configuration interface of the system. This interface is often accessible only during the first moments of the machine start up through the striking of a combination of keys.

The hardware configuration of the machine depends more on how it will be used than on the installed operating system. It is however necessary to specify that deactivation (or activation) of functionalities at the BIOS-level can block the use of these by the system. The technical note « *Hardware configuration recommendations for x86 clients and servers* » [4] discusses the different options that can be found on a contemporary x86 machine.

R9

Hardware configuration

It is recommended to apply the configuration recommendations mentioned in the technical note « *Recommendations for the hardware configuration of client computers and x86 servers* » [4].

4.2 32 or 64 bits Mode

The x86 architecture has evolved over time. Today almost all x86 processors are capable of operating in both 32-bit and 64-bit modes. Other modes (real mode by example) still exist for reasons of backward compatibility. GNU/Linux and Unix distributions for the general public will offer mostly two types of modes:

- the `protected mode`, confined to 32-bit virtual addressing (4 bytes);
- the `long mode`, which allows a wider addressing on 64 bits [1] (8 bytes).

The 64-bit mode has a couple of advantages over the 32-bit mode, despite the absence of some mechanisms allowing the implementation of protections (segmentation in particular):

- relative addressing by register %rip, more efficient for relocatable code like PIC or PIE;
- larger address space, so more bits are accessible for the ASLR;
- memory size that can be allocated to a process is much larger;
- systematic presence of NX/XD protection bits.

R10

32 and 64 bit architecture

When the machine supports it, prefer the installation of a GNU/Linux distribution in 64-bit version instead of 32-bit version.

4.3 IOMMU Service (input/output virtualization)

Activating the IOMMU service protects the system's memory from being arbitrary accessed by devices. The effectiveness of this protection depends greatly on the how the system is built. Even if its design is imperfect (see [11] publication), the IOMMU activation on the system contributes to reducing the risk of arbitrary access to memory from devices.

Activation of the feature will depend on the hardware configuration and the operating system settings: depending on the situation (presence or absence of a functional IOMMU, amount of memory present on the system, etc.) Linux may decide not to initialize the IOMMU. It must therefore be told to force using the IOMMU through a configuration directive passed in parameter at startup.

R11

IOMMU Configuration Guideline

The `iommu=force` directive must be added to the list of kernel parameters during startup in addition to those already present in the configuration files of the bootloader (`/boot/grub/menu.lst` or `/etc/default/grub`).

The configuration change may require running a third-party utility to be taken into account in the configuration (update-grub for example) and the complete reboot of the system. Moreover, the activation of the IOMMU on the system can generate hardware instabilities, it is therefore important to ensure that it functions properly before deploying such a measure to production.

1. Only 48 bits are actually exploited, the unused 16 bits being part of the « non-canonical » addresses.

Listing 4.1: IOMMU activation example

```
# Example for the file '/etc/default/grub'

...
GRUB_CMDLINE_LINUX=" iommu=force"
```

5
System Installation

Each GNU/Linux operating system and distribution adopts its own path during this step. Some configuration elements will however apply almost universally.

Objective

Apply security parameters at first install of the system.

5.1 Partitioning

It is common practice to dedicate partitions to services that can generate a lot of volumetry in order to avoid overloading system partitions. The space to be provisioned for each partition depends on use cases: a file server will need a large volume for `/srv` or `/var/ftp/`, while a log server will be concerned about `/var/log`.

Partitioning must thus make it possible to protect and isolate the various components of the file system. It is by default often unsatisfactory: it does not sufficiently take into account the options `nosuid` (ignore setuid/setgid bits), `nodev` (ignore special character or block files), and `noexec` (ignore execution rights).

It should be noted that depending on the systems and distributions, some of the mounting options will not be applicable transiently; for example utilities, installers or products will estimate that the files written to `/tmp` or `/var` can be executable. In these exceptional cases it is necessary to adapt the partitioning. One of the most frequently encountered situation is that of distributions derived from Debian whose `/var/lib/dpkg` requires execution rights. An alternative is to implement a maintenance procedure during which the updates are installed, just like what is found on other operating systems.

R12

Partitioning type

The recommended partitioning type is as follows:

Mount Point	Options	Description
/	<without option>	root partition, contains the rest of the tree
/boot	nosuid,nodev,noexec (optional noauto)	Contains the kernel and the boot-loader. No access required once the boot finished (except update)
/opt	nosuid,nodev (optional ro)	Additional packages to the system. Read-only editing if not used
/tmp	nosuid,nodev,noexec	Temporary files. Must contain only non-executable elements. Cleaned after reboot or preferably of *tmpfs* type
/srv	nosuid,nodev (noexec,optional ro)	Contains files served by a service like web, ftp, etc
/home	nosuid,nodev,noexec	Contains the *HOME* users. Read-only editing if not in use
/proc	hidepid=2	Contains process information and the system
/usr	nodev	Contains the majority of utilities and system files
/var	nosuid,nodev,noexec	Partition containing variable files during the life of the system (mails, PID files, databases of a service)
/var/log	nosuid,nodev,noexec	Contains system logs
/var/tmp	nosuid,nodev,noexec	Temporary files kept after extinction

The `/boot` contains the boot kernel and the `System.map` file(s) containing the symbol table used by it. This file is often browsed by different malicious programs in order to more easily build « exploits » of kernel code. Ideal partitioning requires that this partition not to be mounted automatically at startup (option `noauto`), and that its mounting is a critical event from a system point of view (for a kernel update or kernel patch for example). But this measure requires adapting system tools to this particular configuration, and therefore requires sharp system expertise during its deployment.

R13

Access Restrictions on the /boot directory

When possible, the `/boot` partition should not be mounted. In any case, access to the `/boot` directory must only be allowed to the `root` user.

It should be noted that some services may still need access to a given tree after a `chroot` operation, without this tree being attached and visible directly from the `chroot` jail. In such cases, the use of *bind* mount points should be considered.

5.2 Selection of packages to install

The packages installation is the crucial step that will determine the set of files that will be present on the system, the services it will provide as well as the packages that will have to be maintained over time.

It is easier to get a minimalist installation by removing all preselected packages, and choose only those necessary for the context of use. For example, running a server does not always require the installation of a local graphical interface (X server).

R14

Installation of packages reduced to the bare necessities

The choice of packets should lead to an installation as small as possible, limiting itself to select only what is required.

Some distributions provide pre-configured « roles ». It is not recommended to base an installation on these roles since the choices of the maintainers of the distribution do not match necessarily the own needs, which will require the installation of additional packages.

R15

Choice of package repositories

Only up-to-date official repositories of the distribution must be used.

R16

Hardened package repositories

When the distribution provides several types of repositories, preference should be given to those containing packages subject to additional hardening measures.

Between two packages providing the same service, those subject to hardening (at compilation, installation, or default configuration) must be preferred.

5.3 Bootloader configuration

For reasons equivalent to the BIOS configuration, the bootloader is an important part of the boot chain. Those of today (GRUB, GRUB 2, etc.) are functionally rich: they allow accessing file systems (and possibly data modifications), booting on USB devices or changing boot options of the selected kernel.

R17

Boot loader password

A boot loader that can set a boot password must be to be privileged. This password must prevent any user from changing the bootloader configuration options.

When the boot loader does not offer the possibility to set a password, a technical measure (and, when appropriate, an organizational measure) must be set up to block any user in his attempts to change the setting.

GRUB and GRUB 2 (bootloaders for x86 architecture) both offer the possibility to set an unlock password. Consult their respective documentation [13] for more information.

5.4 Root password and administrator accounts

The `root` password must be chosen with great care and in accordance with the latest recommendations. His knowledge should be limited to those who need to know it.

R18

Robustness of the administrator password

The `root` password must be robust enough given the recommendations present in the technical note « *Security recommendations for passwords* » [1] available on the ANSSI website.
This password must be unique to each machine.

It is often a common practice to set different levels of privilege on the system based on prerogatives of the administrators. Some may have responsibilities that only concern the web site, others the logging infrastructure, or the databases.

The `root` account is unsuitable for these purposes. It gives full power to the person who has access to it. The use of such a generic account does not facilitate accountability during an incident, and does not promote a model of granular separation of privileges (for example between different system administration teams).

R19

Accountability of administration operations

Each administrator must have a dedicated account (local or remote), and must not use the `root` account as the access account for system administration.

Change of privilege operations must be based on executables that can allow monitoring of the activities performed (for example: `sudo`).

5.5 Installation of additional elements: keys, certificates

The installation stage is an opportunity to set up previously generated elements such as authentication keys or certificates.

R20

Installation of secret or trusted elements

All secret elements or those contributing to the authentication mechanisms must be set up as soon as the system is installed: account and administration passwords, root authority certificates, public keys, or certificates of the host (and their respective private key) among others. If default secret elements are provided, they have to be changed during, or just after, the initial installation of the system.

6
Configuration and system services

The configuration of an operating system is accompanied by the deployment of services. The following recommendations aims to establish some good administration and configuration practices.

Objective

Identify essential goods and appropriate hardening measures to delay as much as possible wide scale compromise of the system.

6.1 Services exposed to uncontrolled flows

Particular attention should be paid to services exposed to uncontrolled flows, that is to say any service communicating with sources that are not trusted (Internet, public Wifi, etc.) or not authenticated. These are the assets which are preferentially attacked and compromised because their access is free.

The terminology « service » is to be taken broadly here. Any flaw or exploitable vulnerability before an authentication step is particularly concerned with this recommendation. For example, a Web service based on basic HTTP authentication may be exploitable on different levels:

1. hardware (firmware, network card drivers, etc.);
2. network (Ethernet, IP, TCP);
3. application (SSL/TLS layer, HTTP headers issued and pre-authentication receipts).

This service must therefore be hardened and monitored, despite the apparent authentication operation performed by the server.

R21

Hardening and monitoring of services subject to arbitrary flows

Services exposed to uncontrolled flows must be monitored and particularly hardened.
Monitoring consists of characterizing the behavior of the service, and reporting all deviation from its nominal operation (this one being deduced from the initial expected specifications).

Hardening is a combination of technical measures that aim to delay or even prevent the compromise of forementioned service. This approach applies from the design phase (privilege separation study, unambiguous specifications, etc.) to the implementation (validation of inputs/outputs, secure configuration, etc.) and the maintenance.

6.2 System configuration - `sysctl`

The *sysctl* are a set of variables that allow you to adapt the settings of the operating system and more particularly its kernel.

This set is constantly enriched according to developments and improvements which are brought to the system. Their scope, use and configuration must therefore be the subject of a regular monitoring if an administrator wants to make the most of their features.

The *sysctl* can act at any level:

- memory: configuration of pagination, allocators, properties of mappings, etc.
- network: IP, TCP, UDP, buffer sizes and characteristics, advanced functions (syn cookies, choice of algorithms, etc.);
- kernel: cache control, swap, scheduling, etc.
- file systems: setuid dumps, hard and soft links, quotas;
- process: allocated resources, execution restrictions, partitioning, etc.

Many *sysctl* are available on a system and their features diverse. It is necessary to refer to their documentation (usually under `doc/Documentation/`, as in the folder tree [14] of the Linux kernel sources) to get a detailed explanation of their role. Some have interesting security features.

R22

Setting up network sysctl

These sysctl are given for a typical host « server » that does not perform routing and having a minimal IPv6 configuration (static addressing). When IPv6 is not used it should be disabled by setting the option `net.ipv6.conf.all.disable_ipv6` to **1**. They are shown here as encountered in the `/etc/sysctl.conf` file:

Listing 6.1: sysctl network settings

```
# No routing between interfaces
net.ipv4.ip_forward = 0
# Reverse path filtering
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.default.rp_filter = 1
# Do not send ICMP redirects
net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.default.send_redirects = 0
# Deny source routing packets
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.default.accept_source_route = 0
# Do not accept ICMPs of redirect type
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.all.secure_redirects = 0
```

```
net.ipv4.conf.default.accept_redirects = 0
net.ipv4.conf.default.secure_redirects = 0
# Log packets with abnormal IPs
net.ipv4.conf.all.log_martians = 1
# RFC 1337
net.ipv4.tcp_rfc1337 = 1
# Ignore responses that do not comply with RFC 1122
net.ipv4.icmp_ignore_bogus_error_responses = 1
# Increase the range for ephemeral ports
net.ipv4.ip_local_port_range = 32768 65535
# Use SYN cookies
net.ipv4.tcp_syncookies = 1
# Disable support for "router solicitations"
net.ipv6.conf.all.router_solicitations = 0
net.ipv6.conf.default.router_solicitations = 0
# Do not accept "router preferences" by "router advertisements"
net.ipv6.conf.all.accept_ra_rtr_pref = 0
net.ipv6.conf.default.accept_ra_rtr_pref = 0
# No auto configuration of prefixes by router advertisements
net.ipv6.conf.all.accept_ra_pinfo = 0
net.ipv6.conf.default.accept_ra_pinfo = 0
# No default router learning by router advertisements
net.ipv6.conf.all.accept_ra_defrtr = 0
net.ipv6.conf.default.accept_ra_defrtr = 0
# No auto configuration of addresses from "routers advertisements"
net.ipv6.conf.all.autoconf = 0
net.ipv6.conf.default.autoconf = 0
# Do not accept ICMPs of redirect type
net.ipv6.conf.all.accept_redirects = 0
net.ipv6.conf.default.accept_redirects = 0
# Deny routing source packets
net.ipv6.conf.all.accept_source_route = 0
net.ipv6.conf.default.accept_source_route = 0
# Maximum number of autoconfigured addresses per interface
net.ipv6.conf.all.max_addresses = 1
net.ipv6.conf.default.max_addresses = 1
```

R23

Setting system sysctl

Here is a list of recommended system sysctl (in format `/etc/sysctl.conf`):

Listing 6.2: List of recommended sysctl

```
# Disabling SysReq
kernel.sysrq = 0
# No core dump of executable setuid
fs.suid_dumpable = 0
# Prohibit unreferencing links to files
# the current user is not the owner
# Can prevent some programs from working properly
fs.protected_symlinks = 1
fs.protected_hardlinks = 1
# Activation of the ASLR
kernel.randomize_va_space = 2
# Prohibit mapping of memory in low addresses (0)
vm.mmap_min_addr = 65536
# Larger choice space for PID values
kernel.pid_max = 65536
# Obfuscation of addresses memory kernel
kernel.kptr_restrict = 1
# Access restriction to the dmesg buffer
kernel.dmesg_restrict = 1
# Restricts the use of the perf system
kernel.perf_event_paranoid = 2
kernel.perf_event_max_sample_rate = 1
kernel.perf_cpu_time_max_percent = 1
```

It is possible to prevent the loading of new modules (including by `root`) through a sysctl. This measure, effective to prevent the modification of the kernel by third-party modules that are potentially malicious, may have non-trivial consequences on the functioning of the rest of the system. It must therefore be ensured that the activation of this option does not lead to significant functional impacts.

This option should be activated once all desired modules have been loaded. It is usually recommended to integrate this option in boot scripts so it is applied as soon as possible, since it protects the kernel from being directly modified by loading modules (this action cannot be reversed, loading new modules will then require a system reboot).

R24

Disabling the loading of kernel modules

The loading of the kernel modules can be blocked by the activation of the sysctl `kernel.modules_disabled`, either directly by the following command (unsaved after reboot):

Listing 6.3: Blocking kernel modules loading (1)

```
# unsaved command after reboot
sysctl -w kernel.modules_disabled=1
```

or by modifying the file `/etc/sysctl.conf`:

Listing 6.4: Blocking kernel modules loading (2)

```
# Prohibition of loading modules (except those already loaded to this point)
# by modifying the file /etc/sysctl.conf
kernel.modules_disabled = 1
```

A security module, Yama, allows to add a sysctl that controls the access rights to the system call `ptrace`. This one is particularly sensitive because it allows to trace the functioning of a process. By default, any user has the right to debug all those belonging to him, which includes processes that store sensitive elements in their memory as keys or passwords (Internet browser, `ssh-agent`, etc.). Compromise of a user process can allow, by rebound, to recover these data.

R25

Yama module sysctl configuration

It is recommended to load the Yama security module at startup (by example passing the `security=yama` argument to the kernel) and configure the sysctl `kernel.yama.ptrace_scope` to a value of at least **1**.

6.3 Access Account Management

6.3.1 Disabling unused user accounts

R26

Disabling unused user accounts

Unused user accounts must be disabled at the system level.

This deactivation goes through the invalidation of the account at the level of his password (remove the `pw_passwd` in the *shadow* and login shell to `/bin/false`).

Listing 6.5: Disabling an unused user accounts

```
# Lock an account
usermod -L <compte>
# Unlocking its login shell
usermod -s /bin/false <compte>
```

The vast majority of Unix/Linux systems isolate each other's applications and services by dedicating each to a specific user account. For example a web server, once started, uses the privileges of a « web » user (called *www* ou *www-data*), while the name server runs under a separate account (for example *named*).

These service accounts are similar to unused user accounts.

R27

Disabling service accounts

Service accounts must be disabled.

Disabling these accounts has no practical effect on the use of the associated credentials (UID). This measure avoids the opening of a user session by a service account. It is important to note that some services can be declared with the `nobody` account. When they are many to adopt such behavior, they find themselves sharing the same account (and privileges) at the operating system level. A web server and a directory using `nobody` can therefore control each other and alter the execution of the other: configuration modification, sending signals, `ptrace` privileges, etc.

R28

Uniqueness and exclusivity of system service accounts

Each service must have its own system account which has to be dedicated to it exclusively.

6.3.2 Expiration time of user sessions

R29

User session timeout

Remote user sessions (shell access, graphical clients) must be closed after a certain period of inactivity.

Most administrators and users often open remote connections to systems they maintain and operate, but may forget to close these sessions. This exposes unnecessarily the host at risks of compromise (reuse of an already open access from a compromised account, session theft, etc.), in addition to unnecessarily consuming machine resources.

This feature is often present in the form of session expiration after idle timeout. Some shells (bash, ksh, zsh) provide a `TMOUT` environment variable that indicates in seconds the maximum idle time for a given shell session.

6.4 PAM and NSS

PAM is a set of modules that allow you to « dynamically » configure the different authentication and account management mechanisms on a GNU/Linux system. The traditional Unix method (*shadow*) is configured through the file `/etc/login.defs`.

NSS is the system layer that handles and queries administrative databases. There are several (`passwd`, `group`, `hosts`, `services`, etc.). Only the management databases of the user accounts will be studied below (those are: `passwd` and `group`).

6.4.1 PAM

PAM's documentation is rich, as are all the features offered by its modules. The purpose of this technical note is not to explain how it works. We assume the administrator to be familiar with this one.

PAM will essentially provide the account management service, that is to say: enable user's authentication, the creation of his session and possibly any operation that must take place when attempting an access: environment creation, ticket or data recovery, access rights, password change, etc.

When a service uses PAM to authenticate a user, the operation is directly performed by a PAM module.

In general, the application submits the authentication elements to the PAM modules. According to their configuration (located in various files present under `/etc/` and `/etc/pam.d/`), PAM returns the result (failure or success) to the application, which will then give access (or not) to system.

Two important elements must be noted:

- PAM modules are called by the application. The application manipulates, at least partially, elements that help authenticate the user (which includes potentially sensitive data such as passwords);
- depending on the PAM modules used, PAM will check the elements through local databases (like `shadow`) or remote databases (LDAP queries, Kerberos, SQL client, etc.).

R30 M I E H

Applications using PAM

The number of applications using PAM must be reduced to what is necessary in order to limit the exposure of sensitive authentication elements.

R31 I E H

Securing PAM Authentication Network Services

When authentication takes place through a remote service (network), the authentication protocol used by PAM must be secure (flow encryption, remote server authentication, anti-replay mechanisms, etc.).

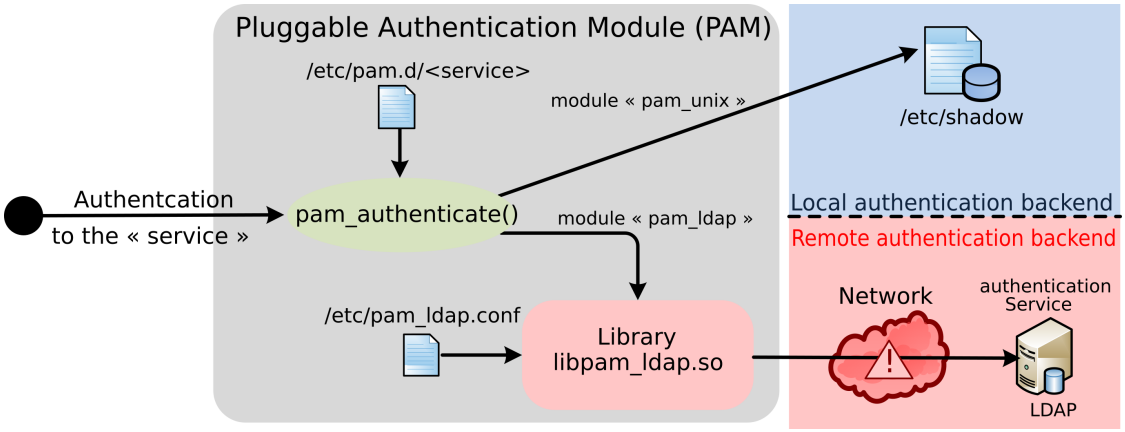


Figure 6.1: Unix PAM + LDAP authentication scheme

Protocols like Kerberos offer such functions (this is the case of `pam_krb5` if the keytab *host* is filled in for the machine). Other modules, such as `pam_ldap` or `pam_mysql`, will not try to establish a secure link between the PAM client and the service if they are not explicitly configured in this way.

In addition to authentication, PAM will allow to load other modules whose features can improve the security of the architecture:

pam_time allows to restrict access to a time slot;

pam_cracklib allows to test the strength of passwords;

pam_passwdqc allows to apply constraints according to a complexity policy passwords (alternative to `pam_cracklib`);

pam_tally allows to temporarily block an account after a defined number of failures;

pam_wheel allows to restrict access to the `root` account to users who are members of a particular group (default `wheel`).

Here are some examples of PAM configuration files for the described modules above. As a reminder, these files are located under `/etc/pam.d/` and have the name of the service they are associated with. Only the configuration directives concerning them are presented. Example with `/etc/pam.d/su` and `/etc/pam.d/sudo`:

Listing 6.6: Blocking root access

```
# Files /etc/pam.d/su and /etc/pam.d/sudo
# Blocks root access to members of the 'wheel' group
auth            required        pam_wheel.so
```

Example with `/etc/pam.d/passwd`:

Listing 6.7: Passwords complexity rules

```
# File /etc/pam.d/passwd
# At least 12 characters, no repetition or monotone sequence,
# 3 different classes (among uppercase, lowercase, numbers, others)
password        required        pam_cracklib.so minlen=12 minclass=3  \
                                        dcredit=0 ucredit=0 lcredit=0 \
                                        ocredit=0 maxrepeat=1         \
                                        maxsequence=1 gecoscheck      \
                                        reject_username enforce_for_root
```

Example with `/etc/pam.d/login` and `/etc/pam.d/sshd`:

Listing 6.8: Automatic accounts blocking

```
# Files /etc/pam.d/login and /etc/pam.d/sshd
# Block the account for 5 min after 3 checks
auth            required        pam_tally.so deny=3 lock_time=300
```

The storage of passwords in plaintext is forbidden because the compromise of the system allows an attacker to reuse them effortlessly towards other services. Their protection must not only rely on access rights to a password database.

R32

Protection of stored passwords

Any password must be protected by cryptographic mechanisms that avoid exposing it in plaintext to an attacker recovering a password database:

- hashing the password with a hash function considered safe (*SHA-256*, with a salt and a fairly large number of turns (65536);
- using a key derivation function on the password and combining it with a hash function considered safe, in accordance with the *Référentiel Général de Sécurité* (a *General Security Framework* providing a regulatory context in France) [7] in order to obtain a valid fingerprint for a given random number ;
- encrypting the password with a secret key (possibly protected through an HSM to which the application is accessing).

PAM and `/etc/login.defs` can be configured to use *SHA-512* according to the recommendations above:

In file `/etc/pam.d/common-password`:

Listing 6.9: Using sha512 (1)

```
# File /etc/pam.d/common-password
password    required    pam_unix.so obscure sha512 rounds=65536
```

In file `/etc/login.defs`:

Listing 6.10: Using sha512 (2)

```
# File /etc/login.defs
ENCRYPT_METHOD       SHA512
SHA_CRYPT_MIN_ROUNDS 65536
```

6.4.2 NSS

Another equally critical element for managing user accounts is the NSS subsystem, which manages all the administrative databases.

When user accounts are stored in an external directory (frequently LDAP), NSS will manage the requests to the directory in order to make the accounts visible from the system.

These requests are anonymous to the directory in the default configuration, with an unprotected communication channel. It is therefore easy for an attacker to retrieve a list of accounts valid from the directory or even spoof the directory server to NSS.

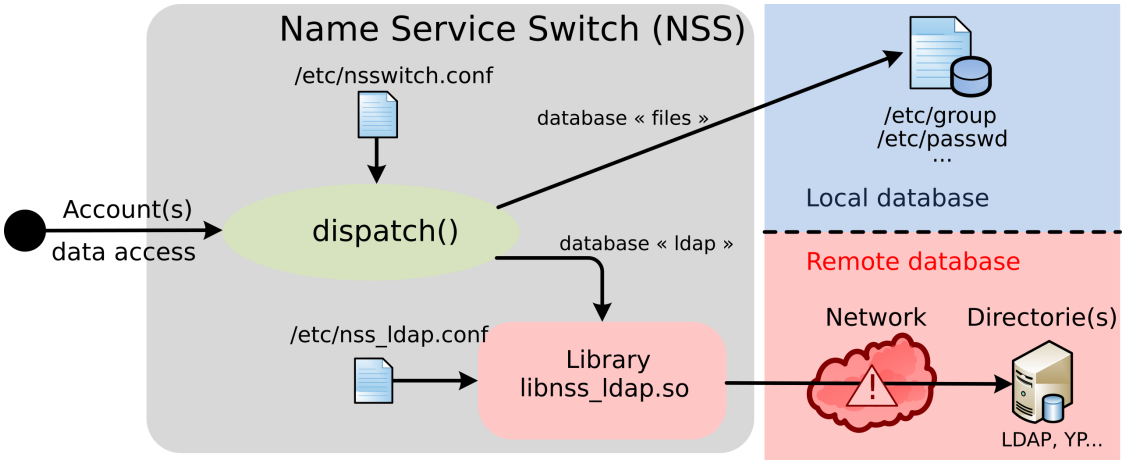


Figure 6.2: NSS Administrative Databases Diagram

R33

Securing access to remote user databases

When the user databases are stored on a remote network service (LDAP type), NSS must be configured to establish a secure link that allows, at minimum, to authenticate the server and protect the communication channel.

R34

Separation of System Accounts and Directory Administrator

It is recommended not to have overlapping accounts between those used by the operating system and those used to administer the directory.

Using directory administrator accounts to perform enumeration queries accounts by NSS must be prohibited.

6.5 Checking file systems and rights

Under Unix, the prevailing model for accessing information by a user is based on the DAC, or discretionary access control. It is the owner of the resource (files, directories, etc.) that specifies the access rights.

6.5.1 `umask`

When the user is not explicit in his request for rights, the system applies a mask (the *umask*), which is by default very permissive (usually `0022`, ie: any file created is readable by all).

R35

umask value

The system umask must be set to `0027` (by default, any created file can only be read by the user and his group, and be editable only by his owner).

The umask for users must be set to `0077` (any file created by a user is readable and editable only by him).

Until recently each GNU/Linux distribution adopted its own way of doing things:

Distribution	Comment
Debian (and derivatives)	the system umask can be directly modified in */etc/init.d/rc* and the users umask in */etc/login.defs*.
CentOS (and derivatives)	the system umask can be directly modified in */etc/sysconfig/init* and the users umask in */etc/login.defs*.

With migrations of many distributions under *systemd*, the system services umask must be indicated directly in the service's configuration files, the default value chosen being `0022` (`UMask` directive). The users umask will have to be specified in */etc/profile*.

Special attention should be paid to files and directories of the following types:

- those containing secret elements, like passwords, password fingerprints, secret or private keys;
- executables with particular bits, like *setuid* or *setgid*;
- the temporary storage directories to which everyone has access;
- named IPC files, such as sockets or pipes, that allow different processes to establish channels of communication between them.

Adequate rights will have to apply rationally to each of these types.

6.5.2 Files with sensitive content

Files containing private keys, secret keys, passwords, fingerprints, etc. (even logs) are considered as sensitive content files.

Even when these files have their contents protected by additional measures (encryption or fingerprints for passwords), it is still necessary to restrict access according to the principle of defense in depth.

R36  Rights to access sensitive content files

Sensitive content files should only be readable by users with strict need to know.

When these files contain passwords (or password fingerprints) they must only be readable by `root`. On the other hand, the public files that contain the list of users are readable by everyone, but are editable only by `root`.

Some examples of files containing sensitive elements:

Listing 6.11: Files containing sensitive element

```
-rw-r----- root root  /etc/gshadow
-rw-r----- root root  /etc/shadow
-rw------- foo  users /home/foo/.ssh/id_rsa
...
```

The analysis scheme is as follows:

1. the sensitive system files must be owned by the `root` account to prevent a rights change to be performed by an unprivileged user;
2. when this file must be accessible to a non `root` user (example: password base for web server), the user associated with the server must be a member of a dedicated group (example: *www-group*) which will have a read-only access to this file;
3. the rest of the users must have no rights.

6.5.3 setuid or setgid executable files

These files are particularly sensitive because they run with the privileges of their owner (or group) owner and not the one of the current user.

R37 

Executable with setuid and setgid bits

Only programs specifically designed to be used with *setuid* bits (or *setgid*) can have these privilege bits set.

The presence of a *setuid* bit (ou *setgid*) on an executable requires that it adopts a certain number of precautions to guard against vulnerabilities related to the change of user. As for example, in a non-exhaustive way: clean your environment and reset a number of elements inherited from the previous context (signaling masks, open file descriptors, etc.). Most executables do not implement such precautions; adding them a *setuid* or *setgid* bit would therefore introduce risks of privileges escalation.

The most common case is *setuid root* executables, which will run with `root` privileges and not those of the calling user. This allows a user to perform operations for which he has *a priori* no right.

In the presence of vulnerabilities, these programs are exploited to provide a `root` shell to a malicious user, or at least to deflect the legitimate use of the program. The *setuid* executables (and incidentally *setgid*) are to be studied on a case by case basis.

R38 

Executable setuid root

setuid executables should be as few as possible. When it is expected that only the administrators of the machine execute them, the *setuid* bit must be removed and it is recommended to prefer using commands like `su` or `sudo`, which can be monitored.

This verification must take place after each update of the system because their rights may have been restored; additional programs may also have been installed during this step. Here is a non-exhaustive list of *setuid root* files that can be encountered. Any executable not mentioned in this list should be examined with particular attention.

Executable	Comment
/bin/mount	To disable, unless absolutely necessary for users.
/bin/netreport	To disable.
/bin/ping6	(IPv6) Same as `ping`.
/bin/ping	(IPv4) Remove setuid right, unless a program requires it for monitoring.
/bin/su	Change user. Do not disable.
/bin/umount	To disable, unless absolutely necessary for users.
/sbin/mount.nfs4	To disable if NFSv4 is unused.
/sbin/mount.nfs	To disable if NFSv2 / 3 is unused.
/sbin/umount.nfs4	To disable if NFSv4 is unused.
/sbin/umount.nfs	To disable if NFSv2 / 3 is unused.
/sbin/unix_chkpwd	Check the user password for non-root programs. To disable if not used.
/usr/bin/at	To disable if *atd* is not used.
/usr/bin/chage	To disable.
/usr/bin/chfn	To disable.
/usr/bin/chsh	To disable.
/usr/bin/crontab	To disable if *cron* is not required.
/usr/bin/fusermount	To disable unless users need to mount partitions FUSE.
/usr/bin/gpasswd	To disable if no group authentication.
/usr/bin/locate	To disable. Replace with `mlocate` and `slocate`.
/usr/bin/mail	To disable. Use a local mailer as `ssmtp`.
/usr/bin/newgrp	To disable if no group authentication.

Executable	Comment
/usr/bin/passwd	To disable, unless non-root users must be able to change their password.
/usr/bin/pkexec	To disable if PolicyKit is not used.
/usr/bin/procmail	To disable unless *procmail* is required.
/usr/bin/rcp	Deprecated. To disable.
/usr/bin/rlogin	Deprecated. To disable.
/usr/bin/rsh	Deprecated. To disable.
/usr/bin/screen	To disable.
/usr/bin/sudo	Change of user. Do not disable.
/usr/bin/sudoedit	Same as `sudo`.
/usr/bin/wall	To disable.
/usr/bin/X	To disable unless the X server is required.
/usr/lib/dbus-1.0/dbus-daemon-launch-helper	To disable when D-BUS is not used.
/usr/lib/openssh/ssh-keysign	To disable.
/usr/lib/pt_chown	To disable (allows to change the owner of PTY before the existence of devfs).
/usr/libexec/utempter/utempter	To disable if the utempter SELinux profile is not used.
/usr/sbin/exim4	Disable if Exim is not used.
/usr/sbin/suexec	Disable if `suexec` Apache is not used.
/usr/sbin/traceroute	(IPv4) Same as `ping`.
/usr/sbin/traceroute6	(IPv6) Same as `ping`.

The following command lists all the setuid / setgid files on the system :

Listing 6.12: List of all the setuid/setgid files on the system

```
find / -type f -perm /6000 -ls 2>/dev/null
```

Removing *setuid* or *setgid* rights is done through the `chmod` command:

Listing 6.13: Removing *setuid* or *setgid* rights

```
chmod u-s <fichier>  # Remove the setuid bit
chmod g-s <fichier>  # Remove the setgid bit
```

6.5.4 Files without user or proprietary group

The files present on the system, whose owner is not part of the administrative `passwd` database must be analyzed and possibly corrected to have a known owner of the system.

The following command lists all files that no longer have a user or associated group:

Listing 6.14: List of all files with no user or associated group

```
find / -type f \( -nouser -o -nogroup \) -ls 2>/dev/null
```

Files without a known owner may be incorrectly assigned to a user when creating his account. We must therefore ensure that no file is in this situation on the system.

6.5.5 Files and folders accessible to all in writing

These directories are in most cases used as temporary storage areas, in which program will record data for processing. Many applications use them incorrectly. Temporary files can then be diverted from their primary function and be exploited as a rebound (for privileges escalations for example).

R39

Temporary directories dedicated to each account

Each user or service account must have its own temporary directory and dispose of it exclusively.

On recent GNU/Linux distributions the most direct way to create a own temporary directory to each user's is to use PAM modules such as `pam_mktemp` or `pam_namespace`.

For various technical reasons, the exclusivity of these temporary files may not be guaranteed. A (imperfect) palliative is then to activate the *sticky bit* on the directory so that only the account that created the file is entitled to delete it.

This prevents a user (or an application) from arbitrarily deciding to delete and replace temporary files belonging to another program.

R40

Sticky bit and write access rights

All directories that are writable by everyone must have the *sticky bit* applied.

This measure is imperfect because it does not prevent racing situations between two programs running simultaneously under the same user account.

The following command lists all the directories that can be modified by everyone and without *sticky bit*:

Listing 6.15: Listing all the directories that can be modified by everyone and without *sticky bit*

```
find / -type d \( -perm -0002 -a \! -perm -1000 \) -ls 2>/dev/null
```

It must also be ensured that the owner of the directory is `root`, otherwise the owner will be able to modify his content despite the *sticky bit*.

The following command lists all the directories that can be modified by anyone and whose owner is not `root`:

Listing 6.16: Listing all the directories that can be modified by anyone and whose owner is not `root`

```
find / -type d -perm -0002 -a \! -uid 0 -ls 2>/dev/null
```

However no regular file needs to be modifiable by everyone. When a file needs to be editable by multiple users or programs at the same time, a group must be created and only this group must have write rights to the said file.

The following command lists all the files that can be modified by everyone:

Listing 6.17: Listing all the files that can be modified by everyone

```
find / -type f -perm -0002 -ls 2>/dev/null
```

6.5.6 Named IPC files, sockets or pipes

Programs can exchange information through `sockets`. Most of the time these sockets are established at the network level (IP). When it comes to establishing connections between local processes, alternatives exist like *pipes* or local Unix *sockets*.

Care must be taken that the access rights that apply to a local socket are those of the directory containing it and not those of the socket itself. Although some systems will honor these permissions, POSIX does not impose it.

R41

Securing access for named sockets and pipes

Named pipes and sockets must be access protected by a directory with appropriate rights.

In particular, local sockets should not be created at the root of a temporary directory accessible in writing to everyone.

Multiple commands allow to get information about sockets in use on the system. According to the applied security policy and the loaded modules, the displayed result may not be exhaustive.

Through `sockstat` (ou `ss`), this command lists all the sockets and gives information of associated processes for local sockets:

Listing 6.18: Listing all the sockets and associated processes for local sockets

```
ss -xp
```

Through `ipcs`, this command lists shared memories. It il also possibe to list their access rights:

Listing 6.19: Listing shared memories and their access rights

```
# List shared memories
ipcs
# List their access rights
ls /dev/shm
```

Through`lsof` to get detailed information about I/O and IPC for system processes:

Listing 6.20: Detailed information about I/O and IPC for system processes

```
lsof
```

6.6 Resident network services

Resident services are all those running on the machine and that can be accessed by a local or remote process.

They are all open doors on the system allowing illegitimate access to it when the service is vulnerable or misconfigured: website to execute arbitrary commands, administrative process that does not use a reliable authentication mechanism, obsolete network service with exploitable vulnerability, etc.

R42

Services and resident daemons in memory

Only network daemons strictly necessary for the operation of the system and the service they render must be resident. They must only listen on the adequate network interfaces.
Other daemons should be disabled and as much as possible uninstalled.

The most common examples are:

- the RPC services (`portmap`, `rpc.statd`, `rpcbind`, etc.) which are in practice used only for an NFS server;
- office services such as `dbus`, `hald`, `ConsoleKit`, `CUPS` or `PolicyKit`;
- the `avahi` service, used for network service publication and discovery;
- the X server, rarely useful on a server;
- additional services whose default configuration is often incomplete: SMTP (`Exim`, `Postfix`), NTP (`ntpd`) and DNS (`Bind`).

The list of resident processes and those listening on the network can be obtained by following commands:

Listing 6.21: List of resident processes and those listening on the network

```
# List of resident processes
ps aux
# List of processes listening on the network
netstat -aelonptu
```

Once the unnecessary services are disabled, the next step is to analyze the last remaining programs by applying the following rules:

1. update of the program;
2. activation of partitioning measures (`chroot`, *containers*, *seccomp* filters, etc.);
3. removal of privileges when those of `root` are not required (by creating an account dedicated to the service and configuring it to use this account);
4. any hardening guidelines documented for the program.

6.7 Tools and monitoring services configuration

Several tools and services can report items of information about the system. A big part of them are not configured optimally after the installation of the system. The following recommendations aim to fill this gap.

6.7.1 syslog

This part complements the technical note *Security recommendations for the implementation of a logging system* [5].

The *syslog* service is the main logging system used under GNU/Linux. He can be broken down into two parts:

- a server (like *syslog-ng* or *rsyslog*) that collects all the system messages in the *syslog* format he receives;
- several clients that send messages to the server, mostly through the `syslog()` API.

A *syslog* server is therefore a unifying element of logging and accesses a large number data from various system sources. Any service or component may request it in order to record a message, without authentication.

R43

Hardening and configuring the syslog service

The chosen *syslog* server must be hardened according to the security guides associated with this server.
The configuration of the service must be performed according to the *Security Recommendations for the implementation of a logging system* [5] accessible on the ANSSI website.

R44

Partitioning the syslog service by `chroot`

When the technical means and its configuration allow it, the *syslog* service must be locked in a `chroot` environment.

R45

Partitioning the syslog service by container

The *syslog* services must be isolated from the rest of the system in a dedicated container.

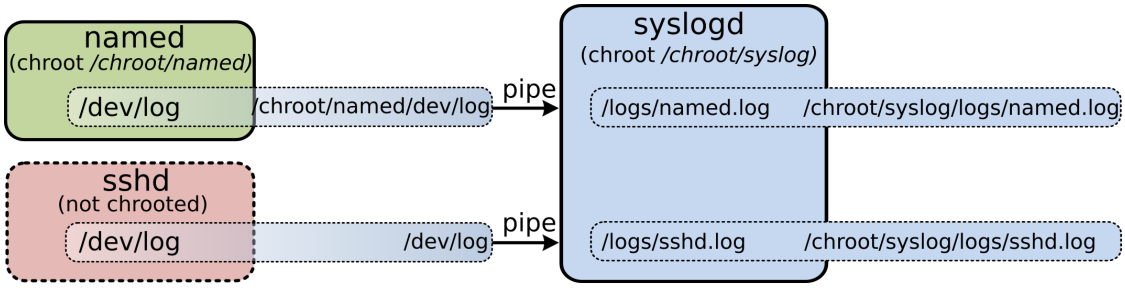


Figure 6.3: *syslog* scheme with *chrooted* and non-*chrooted* services

It is customary to have a resident *syslog* server that only handles messages submitted *locally* by the components of the operating system. The latter will eventually play the role of issuer when these messages must be sent to a log centralization server.

R46

Service Activity Logs

Each service must have a dedicated logging journal on the system. This file should only be accessible through the *syslog* server, and should not be readable, editable or deletable by the service directly.

The goal is to ensure that two levels of protection are respected:

- between services, so that a service can neither manipulate nor access logs recorded by a different service;
- at the level of the service itself, so that in case of compromise, it can not trivially read, erase or alter recorded traces.

The use of the `syslog()` API is a possible solution. Sending messages can be done through the `logger` command in command line.

R47

PDedicated partition for logs

The logs must be in a separate partition from the rest of the system.

The reserved volumetry for logging services is always difficult to assess *a priori*. It is better to isolate the logs from the rest of the volumes on a dedicated partition so that the filling of a partition can not hinder the management of the logs stored there (and reciprocally, that saturation due to the logs can not lead to the operating system unavailability).

6.7.2 Mails and root mails

Messaging is the second main service commonly used by the system to inform on certain evolutions of its state. This usually happens by sending an email to a specific recipient, often a human user.

Depending on the distributions, the installed mail service may be configured as an open relay (accepts any mail that is submitted to it), but with the listening socket only linked to the local loop.

R48

Configuring the local messaging service

When a mail service is installed on the machine, it must be configured so that he accepts only:

- mails to a local user of the machine;
- connections through the local loop (remote connections to the mail service must be rejected).

If possible, prefer to use a light mail redirection service (such as *ssmtp*).

A frequent case is the `cron` service, which systematically submits an email when the executed task displays data on its error output (stderr).

R49

Messaging Aliases for Service Account

For each service running on the machine, as well as the `root` account, an email alias to an administrator user must be configured to receive notifications and reports sent via email.

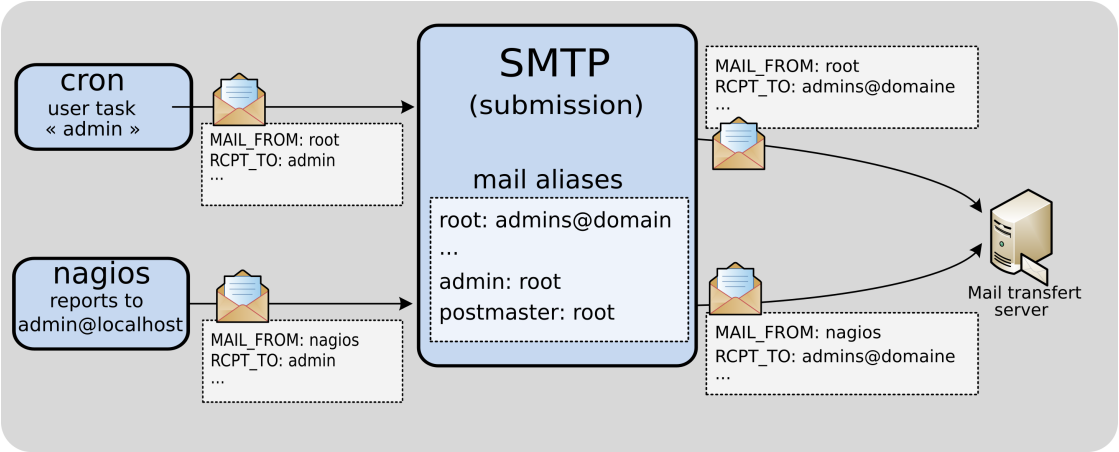


Figure 6.4: Sending alert emails to the administration team

6.7.3 System monitoring with `auditd`

`auditd` is an advanced logging service that is often present on GNU/Linux distributions. It allows to record specific system operations, even to alert an administrator when unplanned privileged transactions take place.

The recorded events depend on the `auditd` rules written. When a registration is triggered, a second service, `audispd`, will take care of its treatment: *syslog* message, sending mail, writing to a file, etc.

`auditd` is able to monitor a large number of actions:

- system calls made;
- access to a particular tree or files;
- module insertions.

See its documentation for more details.

R50

Logging activity by auditd

The logging of the system activity must be done through the `auditd` service.

Here is an example of an `auditd` configuration that seeks to record actions that can be of interest (see file `/etc/audit/audit.rules`):

Listing 6.22: Example of `auditd` configuration

```
# Run of insmod, rmmod and modprobe
-w /sbin/insmod   -p x
-w /sbin/modprobe -p x
-w /sbin/rmmod    -p x
# Log changes in /etc/
-w /etc/ -p wa
# Mounting / unmounting monitoring
-a exit,always -S mount -S umount2
# Suspicious x86 syscalls calls
-a exit,always -S ioperm -S modify_ldt
# Syscalls calls that must be rare and closely monitored
-a exit,always -S get_kernel_syms -S ptrace
-a exit,always -S prctl

# Added monitoring for creating or deleting files
# These rules can have important consequences on the
# system performance
-a exit,always -F arch=b64 -S unlink -S rmdir -S rename
-a exit,always -F arch=b64 -S creat -S open -S openat -F exit=-EACCESS
-a exit,always -F arch=b64 -S truncate -S ftruncate   -F exit=-EACCESS

# Locking the auditd configuration
-e 2
```

The logs thus created by `auditd` can be wordy especially when many activities are reported. The `aureport` tool allows to select interesting information in accordance with very specific criteria: context on authentication failures, event report on specific files or directories, abnormal events (program crash, reconfiguration of network cards), etc.

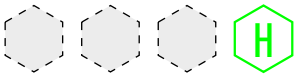
6.7.4 File System Monitoring

The sealing operation of a file system consists of the installation and then the configuration of a service whose function will be to check, at least periodically, the changes made at the level of a file system tree.

This is a feature found within most HIDS. Sealing makes it possible to push up information about the changes (discrepancies between the sealed version and the version currently present on the system).

This is a useful function for administrators. In addition to allowing a periodic audit on the system and reports generation, it allows teams to be informed about changes and thus to obtain a evolution follow-up.

R51



Sealing and integrity of files

Any file that is not transient (such as temporary files, databases, etc.) must be monitored by a sealing program.

This includes among others: directories containing executables, libraries, configura-

tion files, as well as any files that may contain sensitive elements (cryptographic keys, passwords, confidential data).

In addition to the sealing function, some tools are able to browse a file system tree and to identify potentially problematic or incorrect situations (key files or certificates with overly lax rights, password files readable by everyone, log whose size decreases, etc.).

The compromise of a machine may be accompanied by a compromise of the sealing database when it is stored locally, technical measures must be implemented to ensure that the integrity of the database content remains preserved.

R52

Protection of the sealing database

The sealing database must be protected from malicious access by cryptographic signature mechanisms (with the key used for the signature not locally stored in plaintext), or possibly stored on a separate machine of the one on which the sealing is done.

There are many solutions. The most deployed on GNU/Linux systems are Tripwire, Samhain and AIDE.

7
Partitioning and access control solutions

The partitioning is a technique that, when properly implemented, allows to secure the services one against the others while making their use more flexible (multi-instances management, enforcement of restrictions on allocated resources, etc.).

Objective

Deploy partitioning mechanisms to enforce isolation between the processes.

7.1 Partitioning and virtualization

R53

Restricting access of deployed services

The deployed services must have thier access to the system restricted to the strict minimum, especially when it comes to files, processes or network.

It is historically implemented through `chroot` that can restrict the process view of the file system to a given directory (which becomes its root). Nevertheless, access to processes and to the network are not partitioned, the historical Unix model remaining applicable. Nowadays other partitioning solutions exist and offer different levels of abstraction. They are generally characterized by the component that will implement the virtualization interfaces:

- with containers, where the core is able to handle multiple system instances (LXC, Docker, VServer…);
- with emulation, where an emulator reproduces a full physical machine (QEMU, VirtualBox, Parallels);
- with *bare-metal* lightweight hypervisor (Xen, Hyper-V), where the hypervisor will manage (possibly with the help of a host system) different virtual machines;
- with kernel hypervisor (Linux KVM).

These techniques are not exclusive, but have to be used with care, keeping in mind that the interfaces they present are so many doors through which an intrusion is possible: a system based on

containers whose kernel is compromised will see the totality of the containers themselves compromised; the same goes with the virtual machines and their hypervisor.

R54

Virtualization components hardening

Each component supporting the virtualization must be hardened, especially by applying technical measures to counter the exploit attempts.

7.1.1 `chroot`

`chroot` is originally the very first partitioning mechanism which has been used for applications. It is also the poorest. It consists of changing the root directory of a given program: once *chrooted* it cannot access to other directories than the one taken as root and its children. The process has only a partial view of the file system.

`chroot` presents many weaknesses on GNU/Linux, among which:

- inability to forbid a `root` user to escape its jail;
- inability (in the case of some operating systems) to forbid an unprivileged user to escape its jail with the complicity of an external process;
- limited partitioning to the file system; the access to processes, to the network, etc. are not blocked;
- require to have initially root privileges in order to be ran.

R55

`chroot` jail and access right for partitioned service

The `chroot` jail of a service must only contain the strict minimum to run the service properly.

When locked in the `chroot` jail, the service must always run with the privileges of a normal, dedicated user (non `root`), dedicated user (whose identity is used uniquely by the jail) and with no write access to this new root.

The administrator wishing to use a `chroot` jail must be careful in its implementation: it must be ensured that the partitioned process cannot access in read/write directly to the root. It would able it to create and then control files whose path is sensitive (as ./etc/shadow), that would be potentially used wrongly by any binary that would end up locked in this jail.

`chroot` is most often implemented as an internal mechanism of protection for a given service: once the most privileged operations are executed, the service uses `chroot` to change its root, then change to a non-privileged user in order to lose its `root` administrator rights.

R56  Enablement and usage of `chroot` by a service

`chroot` must be used and enabled when the service supports this mechanism.

7.2 Access control and advanced security mechanisms

The access control consists in ensuring that an entity (process or user) has necessary and sufficient rights to access to a given resource. Although an access control allows also to enforce the partitioning, the chosen approach is generally different from the one adopted through virtualization mechanisms: the access control often leaves the reference to a system object visible to the application and returns an error in case of insufficient privilege to access it while a system based on virtualization will isolate the application by the lack of reference to this object (pointer, access path,etc.).

Historically the access control on Unix/Linux is based on the concept of user. Other methods are applicable nowadays through LSMs, whose `SELinux` and `AppArmor` are the most known and used. It is important to note that these access models come in addition to the traditional Unix user model and do not replace it. Moreover `AppArmor` and `SELinux` are mutually exclusive: the two systems cannot run together on the same Linux kernel.

7.2.1 Traditional UNIX model

The historical access model is based on the concept of users who are recognized by the system through unique identifiers (UID). Each process, file, directory, resource, etc. is associated to a UID (its owner), which coupled with rights will allow or deny an access to a resource.

Several UID can be grouped. A unique identifier can be associated to this group: a GID. The principle of access management remains however similar to the UID.

This access model is a DAC (for discretionary access control). It is discretionary because the user owner of a resource is the one who specifies its access rights, with read, write and execute rights each given for:

- the user owning the resource;
- the group owner of the resource;
- the rest of the users.

This approach is still common today and remains the one used by default on GNU / Linux. It presents however important limitations:

- the rights are at the owner's discretion, which may not be suitable for environments restricted by a security policy;
- the owner may fail to provide the proper rights to its resources, which offers access opportunities to confidential data (or even to compromise the system);
- the isolation between users is coarse, the default rights are often too laxist;
- there are only two levels of privileges on the system: `root` administrator, and the normal users, without privilege. The change of user requires the use of sensitive binary on the system (those with *setuid root*);
- it does not allow to withdraw special privileges to a process according to the context: the web browser and the word processor from the same user will have as many privileges on the resources;
- the attack surface is large, a standard user has access to a lot of information on the underlying operating system.

These limitations have contributed to the emergence of rights delegation management tools, the most widely known being `sudo`. Other approaches, such as the LSM, offer more possibilities, but at the price of a higher investment and complexity.

7.2.2 sudo

`sudo` is an utility installed when there is a need to delegate rights and privileges to different users. This delegation is based on the possibility for a given user to run a command previously defined with the privileges of another user. In order to achieve this, `sudo` is an executable *setuid root*. It is important to be concerned about its security at two levels:

- at the hardening level, to prevent a malicious user to exploit the binary vulnerabilities;
- at its configuration level, when giving the right to a user to perform specific commands may give him more privileges and prerogative than initially necessary.

The sudo operation relies heavily on the traditional Unix model to work, which is enriched with a finer transition logic than the one originally allowed by su. The items below are intended to give some recommendations and ideas to explore in order to use and to configure sudo so that it offers the least possible circumvention.

These recommendations apply equally for cases where sudo is used for the execution of privileged commands with a normal user, or the restriction of privileges when a privileged user (such as root) wants to run a command with lower rights, as a precautionary measure.

Access rights sudo is usually installed by default with open rights and allows any user to use it (execution right for everyone). sudo being a privileged executable and complex by its configuration, it is better to restrict its execution rights to a dedicated user group. This reduces the attack surface, especially when the system and the binary itself are victims of exploitable vulnerabilities by any user (CVE-2012-0809, CVE-2012-0864).

R57

Group dedicated to the use of sudo

A group dedicated to the use of sudo must be created. Only the members of this group should have the right to run sudo.

A possible case is to create a dedicated group (here: sudogrp), which is assigned the rights to run sudo. Only the members of this group will be able to call it:

Listing 7.1: Dedicated sudo group

```
# ls -al /usr/bin/sudo
-rwsr-x---. 2 root sudogrp [...] /usr/bin/sudo
```

This modification must be checked and possibly applied after each update, as these modifications may be overwritten by the installation scripts.

General configuration The configuration is done by editing the file /etc/sudoers with the visudo command. It contains a set of guidelines that will allow sudo to know the commands a user is allowed to run or not, this may be based on the machine hostname.

It is particularly difficult to provide a set of recommendations that can cover all situations where sudo can be used. Indeed, the number of available commands, the architecture of the information system, the installed services and their configurations, makes the establishment of a secure standard configuration almost impossible.

Some good practices must be followed in order to avoid, as much as possible, configuration errors that could lead to circumventions of the security policy. The reading of the `sudoers` man page is highly recommended, especially the part dealing with shell escapes.

R58

Sudo configuration guidelines

The following guidelines must be enabled by default:

noexec	applies the **NOEXEC** tag by default on the commands;
requiretty	requires the user to have a tty login;
use_pty	uses a pseudo-tty when a command is executed;
umask=0027	forces umask to a more restrictive mask;
ignore_dot	ignores the ' . ' in `$PATH`;
env_reset	resets the environment variables;
passwd_timeout=1	allocates 1 minute to enter its password.

Resulting in the `/etc/sudoers` file:

Listing 7.2: Guidelines for the `/etc/sudoers` file

```
Defaults noexec,requiretty,use_pty,umask=0027
Defaults ignore_dot,env_reset,passwd_timeout=1
```

The remaining of the file is then predominantly composed of rules allowing to declare, for a given user (or group), the list of commands he is allowed to execute, all helped with alias specifications when the right management policy becomes complex (`User_Alias`, `Runas_Alias`, etc.). The verification of the execution right (or not) of a command is based on a string comparison between the command desired by the user and the specification present in `sudoers`. The simplest model is the following:

Listing 7.3: String template for the verification of a command execution rights

```
utilisateur   hostname = ( utilisateur-cible ) commande, [...]
```

R59

User authentication running sudo

An authentication of the calling user must be performed before any command execution with `sudo`.
The keyword `NOPASSWD` must not be used.

The authentication requirement permits to delay a clear escalation of privileges when an account is compromised, especially when the attacker is not able to authenticate as the legitimate user. For convenience `sudo` is not requesting the password again after a configurable period of time (15 minutes by default). In all cases, an authentication must have been done initially.

R60

Privileges of target sudo users

The targeted users of a rule should be, as much as possible, non privileged users (i.e.: non `root`).

It is common that the command to be executed does not require superuser rights (editing a file whose the owner is not `root`, sending a signal to an unprivileged process,etc.). In order to limit any attempt of privilege escalation through a command, it is better to apply normal user rights.

R61

Limiting the number of commands requiring the use of the EXEC option

The commands requiring the execution of sub-processes (*EXEC* tag) must be explicitly listed and their use should be reduced to a strict minimum.

Functionally rich commands can be executed via `sudo`, like text editors (`vi`) or binaries (`tcpdump`). They may allow execution of sub-commands in the environment created by `sudo` and thus enable a user to run other programs with privileges not foreseen initially. This may well lead to circumventions of the security policy, or even privilege escalations.

To this end, `sudo` can override the different functions allowing to execute other programs. This allows for example to block the trivial creation of a sub-shell through `vi`. It is however important to note that this protection is not perfect and only override the functions allowing to create these new processes. The process remains free to execute the system calls it wants (including `execve`) and so bypass this protection mechanism.

R62

Good use of negation in a sudoers file

Policies applied by sudo through the sudoers file should not involve negation.

Specifying access right using negation (blacklist approach) is inefficient and can be easily circumvented. For example, it is expected that a specification like 7.4 prevents the execution of the shell but that's not the case: just copy the binary `/bin/sh` to a different name to make it executable again through the rule keyword *ALL*.

Listing 7.4: sudo rule to avoid

```
# To avoid absolutely, this rule can be easily circumvented!
user    ALL = ALL,!/bin/sh
```

To determine whether a command is executable by a user, `sudo` performs a string comparison between the desired command and the corresponding specifications in the `sudoers` following the rules of the *globbing* shell; this evaluation includes the arguments.

R63 

Explicit arguments in sudo specifications

All commands in the `sudoers` file must strictly specify the arguments allowed to be used for a given user.
The use of `*` (*wildcard*) in the rules should be avoided as much as possible. The absence of arguments in a command must be specified by the presence of an empty chain (“”).

Any argument can modify quite significantly the behavior of a program, whether regarding the realized operation (read, write, delete, etc.) or accessed resources (path in a file system tree). To avoid any possibility of misuse of a command by a user, the ambiguities must be removed at the level of its specification.

On some systems, the kernel messages are only accessible by `root`. If a user nevertheless must have the privileges to read them, the argument of the dmesg command has to be restricted in order to prevent the user from flushing the buffer through the `-c` option:

Listing 7.5: Do not allow flushing the buffer

```
user    ALL = dmesg ""
```

It is important to note that allowing an unprivileged user to run under the `root` identity a program susceptible of making an arbitrary write (made possible, by writing too latitudinarian rule) is somehow giving `root` privileges to this user because this one could, by different methods (handling of password files, modification of *setuid root* binary, etc.) obtain the `root` privileges without the access control and enforcement that `sudo` allows.

It is quite common to allow a user to edit a file through `sudo` by calling directly a text editor. A text editor is a program functionally rich, that can open and edit other files even by internal commands or even run a shell script. Some editors may offer a full privileged environment to a user. `sudoedit` solves these problems by editing a temporary version of the file with the current user rights, then replacing the original file by this temporary version once the operation is finished. The editor runs thus only with the current user rights and never with the privileges of the target user.

R64  Good use of sudoedit

A file requiring `sudo` to be edited, must be edited through the `sudoedit` command.

7.2.3 AppArmor

AppArmor is one of the LSM frequently encountered on GNU/Linux, used by default by variants of SUSE (Including OpenSUSE) and Ubuntu. It is a MAC model where an authority decides accesses allowed to an application without giving it the capability to alter them.

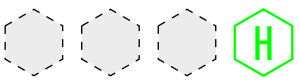
Its documentation is directly accessible on the project website [9]. Its configuration is based on specifications of rights based on paths in the file system tree, and applied for each executable.

In addition to the access specifications, AppArmor allows to block the use of POSIX capabilities, and also to restrict the network use (`network` directive). These restrictions, however, remain basic and do not have the richness of `iptables` rules [2].

When an executable under AppArmor calls another executable, its security profile allows to declare how the transition should take place (inheritance of the current profile, activation of another security profile, exit from confinement, etc.)

Although simple to understand for an administrator accustomed to the traditional access Unix model, AppArmor does not allow to attach security labels to a flow or messages. It is a framework that essentially focuses on the access and privileges specification for executables, the concept of a security label is absent.

Many of the GNU/Linux distributions using it provide, at least for most sensitive executables (`syslogd`, `ntpd`, *setuid root* binaries…) default AppArmor profiles in the directory `/etc/apparmor.d/` (with a file per executable). Operations (including denied access) are recorded by `auditd` directly in the file `/var/log/audit/audit.log`.

R65  Enable AppArmor security profiles

All AppArmor security profiles on the system must be enabled by default.

2. System utility allowing to configure the firewall rules on GNU/Linux.

Activation of AppArmor is different depending on the distributions, and is usually based on running a script like `/etc/init.d/apparmor`. To make sure AppArmor is active, use `aa-status` once the startup is complete:

Listing 7.6: Checking AppArmor activation

```
# aa-status
apparmor module is loaded.
5 profiles are loaded.
5 profiles are in enforce mode.
...
4 processes have profiles defined.
4 processes are in enforce mode.
...
0 processes are unconfined but have a profile defined.
```

This includes checking that the processes running on the system, and for which a profile is declared, are confined by AppArmor.

7.2.4 SELinux

SELinux (for *Security-Enhanced Linux*) is a widely used LSM in the Red Hat and CentOS distributions.

SELinux relies on the use of *labels* assigned to objects (files, processes, sockets, etc.) that allow, depending on the *domain* to which a process is confined, to grant or deny access to a resource.

At the file system level, labels are stored as extended attributes and can be obtained via `ls -Z`. Those processes can be displayed with `ps -Z`.

The use of labels enables to declare areas of responsibility and therefore to know at every moment the operations that a process can perform on a given resource. SELinux also allows to define *transitions* from one domain to another when a user requires specific access privileges. The granularity of access is much finer than in AppArmor, at the cost of a greater complexity.

The SELinux documentation is extensive and accessible on the project's wiki webpage [17].

This section is limited to giving some checks so that any administrator wishing to use the profiles provided by its distribution can do it correctly. The SELinux policies available by default on distributions are:

minimum	(*minimum*) policy which confines only a very limited number of services ;
targeted	(called *targeted* or *default*) policy partitioning each system service into a domain. Interactive users are not partitioned;
multi-level	(*mls*) policy including targeted policy and which additionally proposes the use of hierarchical classification levels (sensitivity).

The use of *minimum* policy is not recommended because it does not confine system services in different domains. According to their creators, this policy shoud only be used on constrained environments or for testing purposes [16].

The *mls* policy was created to manipulate files with different sensitive levels on a single system, respecting the *need-to-know* principle. It's worth noticing this policy *does not allow to handle sensi-*

tive information on systems with lower or incompatible sensitivity level (please refer to [6] for further information). This policy will not be detailled hereafter.

The rest of this section will focus on the default *targeted* policy. Each system service (daemon) is confined to a separate domain. Multiple instances of the same service running on one single domain can be separated using the categories. Interactive users are by default associated with an unconfined domain (*unconfined_t*) that does not impose additional restrictions. In this case, the access control model is very close to the DAC.

R66

Enabling SELinux targeted Policy

It is recommended to enable SELinux in *enforcing* mode and to use the *targeted* policy when the distribution supports it and that it does not operate another security module than SELinux.

Activating the *targeted* policy typically involves five steps:

1. Check the value of `SELINUX` and `SELINUXTYPE` variables in the `/etc/selinux/config` file:

   ```
   # grep ^SELINUX /etc/selinux/config
   SELINUX=enforcing
   SELINUXTYPE=targeted
   ```

2. If the value of `SELINUX` is not *enforcing* or if the value of `SELINUXTYPE` is not *targeted* (or *default*), then edit the file and change the variables values;

3. If SELinux is not in *enforcing* mode, it is recommended to check that SELinux labels are well attached to the files on the whole system. This verification can be scheduled so it is executed automatically at next boot using the following command:

   ```
   # fixfiles onboot
   ```

4. If the value is not *targeted* (or default), edit the file to declare the variable `SELINUXTYPE` to *targeted* then reload the security policy:

   ```
   # semodule -R
   ```

5. It can be necessary to reboot the system so the policy change or SELinux activation are well taken into account;

6. Then make sure that SELinux is active and with the right parameters:

   ```
   # sestatus
   SELinux status:                 enabled
   ...
   Loaded policy name:             targeted
   Current mode:                   enforcing
   ...
   ```

The behavior of a security policy can be modified through Boolean variables. Some are useful from a security point of view.

R67

Setting SELinux booleans

It is recommended to set the following Boolean values:

- **allow_execheap** (forbid processes to make their heap executable): *off*;
- **allow_execmem** (forbid of processes to have both write and execute rights on memory pages): *off*;
- **allow_execstack** (forbid processes to make their stack executable): *off*;
- **secure_mode_insmod** (prohibits dynamic loading of modules by any process): *on*;
- **ssh_sysadm_login** (forbid SSH logins to connect directly in sysadmin role): *off*.

The default value of these variables can be modified with the `setsebool` command:

Listing 7.7: `setsebool` usage

```
setsebool -P allow_execheap=off
setsebool -P allow_execmem=off
setsebool -P allow_execstack=off
setsebool -P secure_mode_insmod=on
setsebool -P ssh_sysadm_login=off
```

There are many others. The commands `getsebool -a` or `semanage boolean --list` enable to obtain information about Boolean variables defined in the policy used on a system. A big part of these variables are also documented in the CentOS wiki [15]. Like AppArmor, the operations are recorded by `auditd` directly in the file `/var/log/audit/audit.log` under the keyword AVC.

R68

Uninstalling SELinux Policy Debugging Tools

SELinux policy manipulation and debugging tools should not be installed on a machine in production.

Their use must be limited to the development machines, on which the interpretation of an error reported in SELinux audit logs can be performed.

The *setroubleshootd* daemon must be disabled and the following packages uninstalled: setroubleshoot, setroubleshoot-server, setroubleshoot-plugins.

R69

Confining interactive non-privileged users

Interactive non-privileged users of a system must be confined by associating them with a SELinux confined user.

By default, interactive user rights are not restricted in the *targeted* policy. But it is possible to selectively confine users who do not need to perform administrative tasks on a system by associating them with the SELinux *user_u* user by the help of the following command:

Listing 7.8: Confining a user with SELinux

```
usermod -Z user_u <user>
```

Recommendation List

Bibliography

[1] *Recommandations de sécurité relatives aux mots de passe*.
Note technique DAT-NT-001/ANSSI/SDE/NP v1.1, ANSSI, juin 2012.
https://www.ssi.gouv.fr/mots-de-passe.

[2] *Recommandations pour la sécurisation des sites web*.
Note technique DAT-NT-009/ANSSI/SDE/NP v1.1, ANSSI, août 2013.
https://www.ssi.gouv.fr/securisation-sites-web.

[3] *(Open)SSH Secure use recommendations*.
Note technique DAT-NT-007-EN/ANSSI/SDE/NP v1.2, ANSSI, août 2015.
https://www.ssi.gouv.fr/nt-ssh.

[4] *Recommandations de configuration matérielle de postes clients et serveurs x86*.
Note technique DAT-NT-024/ANSSI/SDE/NP v1.0, ANSSI, mars 2015.
https://www.ssi.gouv.fr/nt-x86.

[5] *Recommandations de sécurité pour la mise en œuvre d'un système de journalisation*.
Note technique DAT-NT-012/ANSSI/SDE/NP v1.0, ANSSI, décembre 2013.
https://www.ssi.gouv.fr/journalisation.

[6] *Instruction générale interministérielle n°1300*.
Référentiel Version 1.0, ANSSI, novembre 2011.
https://www.ssi.gouv.fr/igi1300.

[7] *Référentiel général de sécurité (RGS)*.
Référentiel Version 2.0, ANSSI, juin 2012.
https://www.ssi.gouv.fr/rgs.

[8] *Recommandations pour la mise en place de cloisonnement système*.
Guide ANSSI-PG-040 v1.0, ANSSI, décembre 2017.
https://www.ssi.gouv.fr/guide-cloisonnement-systeme.

[9] *AppArmor documentation*.
Web page, GitLab B.V.
https://gitlab.com/apparmor/apparmor/wikis/Documentation.

[10] *French CERT Website (in French)*.
Web page, ANSSI.
http://www.cert.ssi.gouv.fr/.

[11] *Analysis of IOMMU service efficiency (in French)*.
Vincent Nicomette Eric Lacombe, Fernand Lone Sang and Yves Deswarte.
Scientific publication, june 2010.
https://www.sstic.org/2010/presentation/Analyse_de_l_efficacite_du_service_fourni_par_une_IOMMU/.

[12] *Licence ouverte / Open Licence*.
Page Web v2.0, Mission Etalab, avril 2017.
https://www.etalab.gouv.fr/licence-ouverte-open-licence.

[13] *GRUB Website*.
Web page.
http://www.gnu.org/s/grub/.

[14] *Linux kernel sources documentation*.
Web page.
https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/Documentation.

[15] *Linux CentOS booleans documentation*.
Web page.
https://wiki.centos.org/fr/TipsAndTricks/SelinuxBooleans.

[16] *Fedora project SELinux policies website*.
Page web.
https://fedoraproject.org/wiki/SELinux/Policies.

[17] *SELinux project Web page*.
Web page.
http://selinuxproject.org/page/Main_Page.

ANSSI-BP-028-EN
Version 1.2 - 22/02/2019

AGENCE NATIONALE DE LA SÉCURITÉ DES SYSTÈMES D'INFORMATION

ANSSI - 51, boulevard de La Tour-Maubourg, 75700 PARIS 07 SP
www.ssi.gouv.fr / conseil.technique@ssi.gouv.fr

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE
Premier ministre
SGDSN