Resumen Teórica 18 : Mecanismos de Protección

Tomás F. Melli

October 2025

Índice

1 Introducción **3**
1.1 Control del entorno de ejecución . . . 3
1.1.1 Chroot – UNIX Standard Jail . . . 4
1.1.2 FreeBSD Jail . . . 4

2 OpenSSH Privilege Separation **4**

3 Sandboxing **5**
3.1 Java Sandbox (deprecado) . . . 6

4 BSD Security Levels **7**

5 Seguridad en Windows: Control de Cuentas de Usuario y Mecanismos de Integridad **7**
5.1 Windows User Account Control (UAC) . . . 8
5.2 Windows Integrity Control . . . 8
5.3 Servicios y mínimos privilegios en Windows 2003 . . . 8
5.4 Mejoras en Windows 2008 . . . 9
5.5 Windows AppLocker . . . 9
5.6 Windows 2012 y 2016 . . . 9
5.7 Active Directory . . . 10
5.8 Políticas de Grupo y Plantillas de Seguridad en Windows . . . 11
5.9 Microsoft LAPS (Local Administrator Password Solution) . . . 12

6 Linux Capabilities y Sudo **13**

7 Máquinas Virtuales y Contenedores **14**
7.1 Virtual Machine . . . 14
7.1.1 Hypervisor . . . 14
7.2 Contenedores . . . 15
7.3 Docker y el ecosistema de contenedores . . . 16
7.3.1 Arquitectura interna de Docker . . . 16
7.4 Seguridad en Máquinas Virtuales y Contenedores . . . 17
7.4.1 Secure Supply Chain . . . 18
7.4.2 Software Defined Networks (SDN) . . . 18
7.4.3 Trusted Execution Environment (TEE) . . . 18

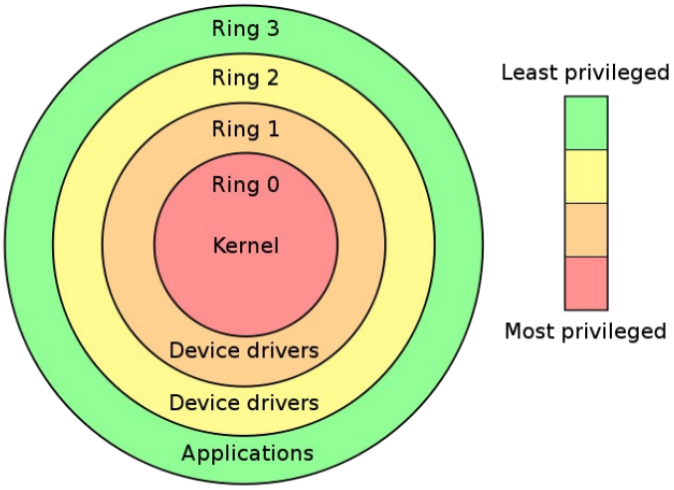
8 Security-Enhanced Linux (SELinux) **18**
8.1 Modelos de Control de Acceso . . . 19
8.2 Funcionamiento Interno de SELinux . . . 19
8.2.1 Implementación en el Núcleo . . . 20
8.2.2 Dominios y Confinamiento . . . 20
8.3 Contextos de Seguridad . . . 21
8.4 Mecanismos de Control en SELinux . . . 21
8.5 Type Enforcement (TE) . . . 21
8.5.1 Decisiones y Transiciones . . . 21
8.5.2 Toma de Decisiones . . . 22
8.5.3 Implementación . . . 22

8.5.4 Componentes del Sistema . . . 23
8.5.5 Dominios Predefinidos y Evolución . . . 23
8.5.6 Archivos de Configuración . . . 23
8.5.7 Modos de Operación . . . 24
8.6 Comandos y Utilitarios de SELinux . . . 24
8.6.1 Comandos Modificados . . . 24
8.6.2 Utilitarios SELinux . . . 24
8.6.3 Manejo de Booleans y Contextos de Archivos . . . 24
8.6.4 Auditoría y generación de políticas . . . 25
8.6.5 Httpd y Booleans específicos . . . 25
8.7 SELinux GUI y Cambio de Contexto de Archivos . . . 25
8.7.1 SELinux GUI – system-config-selinux . . . 25
8.7.2 Cambio de Contexto de Archivos . . . 25
8.7.3 Shellshock . . . 26
8.7.4 Por qué es peligroso? . . . 27

1 Introducción

X86 Privilege Levels

Los procesadores **x86** implementan un modelo jerárquico de privilegios conocido como *rings* (anillos de protección), diseñado para aislar el código del sistema operativo del de las aplicaciones y garantizar la seguridad del sistema. En este esquema, el hardware define **cuatro niveles de privilegio** numerados del **0 al 3**:



- **Ring 0:** máximo privilegio, normalmente reservado para el *kernel* del sistema operativo.
- **Ring 3:** menor privilegio, usado por las *aplicaciones de usuario*.
- **Rings 1 y 2:** existen a nivel arquitectónico, pero **no suelen utilizarse** en sistemas operativos modernos.

En la práctica, los sistemas x86 típicos (como Linux o Windows) solo emplean **dos niveles**: **Ring 0 (modo kernel)** y **Ring 3 (modo usuario)**. Esta simplificación reduce la complejidad y mejora la eficiencia del cambio de contexto entre modos.

Comparación con ARM y x64

- En **x64 (AMD64)**, el modelo de protección se simplifica aún más: aunque hereda la noción de *rings* de x86, los sistemas operativos modernos siguen utilizando solo **dos niveles** (kernel/usuario). Además, x64 introduce el *modo de ejecución largo (long mode)* y extensiones como *System Management Mode (SMM)* y *Virtual Machine Extensions (VMX)*, que añaden nuevos contextos privilegiados fuera de los anillos clásicos.
- En **ARM**, los niveles de privilegio están organizados en *Exception Levels (EL0–EL3)*:
  - **EL0:** espacio de usuario
  - **EL1:** sistema operativo
  - **EL2:** hipervisor (virtualización)
  - **EL3:** firmware o monitor seguro (*TrustZone*)

  A diferencia de x86, ARM utiliza activamente **más de dos niveles**, lo que le permite un control más granular en entornos de virtualización y seguridad embebida.

1.1 Control del entorno de ejecución

En los sistemas UNIX y derivados, el control del entorno de ejecución de los procesos es un aspecto clave de la **seguridad del sistema**. Una de las técnicas más antiguas y simples para limitar el alcance de un proceso es el uso de **chroot**, conocido como *UNIX standard jail*. El concepto fue introducido para permitir que un proceso vea y opere dentro de un subconjunto del sistema de archivos, funcionando como una "jaula" (*jail*) que restringe su acceso a recursos externos.

1.1.1 Chroot – UNIX Standard Jail

El mecanismo **chroot** (change root) consiste en **cambiar la raíz del sistema de archivos visible** para un proceso y sus hijos. Esto significa que todas las operaciones de archivos —como `open()`, `read()`, `write()` o `exec()`— realizadas sobre el directorio `/` dentro del entorno *chroot* se corresponden con operaciones sobre un **subdirectorio del sistema de archivos real**. De esta forma, el proceso "cree" estar en el sistema completo, cuando en realidad está limitado a un árbol de directorios particular.

- **Objetivo:** aislar procesos del resto del sistema, impidiendo el acceso a información sensible o configuraciones críticas.
- **Implementación:** el sistema redirige todas las referencias al directorio raíz (`/`) hacia un subdirectorio que actúa como entorno virtual.
- **Ventajas:** simplicidad, bajo costo de implementación, útil para entornos de prueba, servidores FTP o compilación de paquetes.

Sin embargo, el aislamiento de `chroot` es **parcial**:

- No restringe todas las *system calls* del proceso; por ejemplo, un proceso aún puede listar o interactuar con otros procesos del sistema mediante interfaces del kernel.
- Si el proceso dentro de la jaula tiene privilegios de `root`, puede **salir de la jaula** utilizando llamadas al sistema o montando nuevos dispositivos.

Por estos motivos, `chroot` no debe considerarse un mecanismo de seguridad completo, sino una herramienta de contención básica o de conveniencia administrativa.

1.1.2 FreeBSD Jail

Para superar las limitaciones de `chroot`, el sistema operativo **FreeBSD** introdujo la tecnología **Jail**, una extensión más potente y segura del concepto. Cada *Jail* proporciona un entorno de ejecución virtualizado y completamente aislado del resto del sistema.

- Cada **Jail** ofrece su propio espacio de archivos, procesos, usuarios y administrador, de modo que desde el interior, el entorno se percibe como un sistema independiente.
- Las **Jails** están completamente separadas entre sí: los procesos dentro de una jail no pueden ver ni interactuar con los de otra.
- Permiten la **delegación de tareas** o servicios (como servidores web o DNS) a administradores secundarios sin otorgarles control total sobre el sistema anfitrión.

Este enfoque convierte a las *FreeBSD Jails* en una forma temprana de virtualización ligera, antecedente directo de tecnologías modernas como los **Solaris Containers** y los **Linux VServer**. Su diseño está orientado a la **seguridad y control de aislamiento**, proporcionando una solución robusta para ejecutar servicios en entornos compartidos sin comprometer la integridad del sistema principal.

2 OpenSSH Privilege Separation

En los sistemas modernos, los servicios de red deben minimizar la superficie de ataque reduciendo la cantidad de código que se ejecuta con privilegios elevados. El mecanismo de **Privilege Separation** introducido en **OpenSSH** es una técnica de diseño orientada a mejorar la seguridad del servicio `sshd`, separando las tareas privilegiadas de las no privilegiadas en diferentes procesos con distintos niveles de permiso.

Concepto de Privilege Separation

El principio fundamental de la *Privilege Separation* consiste en dividir el servidor SSH en **dos procesos distintos** con funciones claramente separadas:

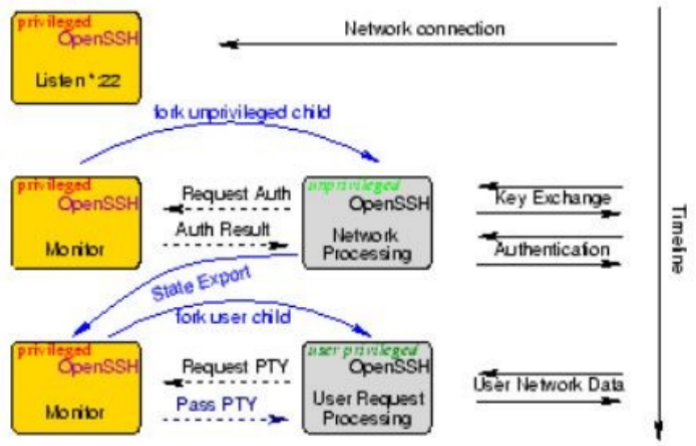
- Un **proceso privilegiado** (padre), que mantiene los permisos de `root` y se encarga exclusivamente de operaciones sensibles, como:
  - Escuchar en el puerto 22 (requiere privilegios).
  - Autenticar usuarios y verificar credenciales.

- Configurar sesiones y asignar recursos del sistema.

- Un **proceso sin privilegios** (hijo), que ejecuta el código de red principal y maneja la comunicación con el cliente. Este proceso se ejecuta en un entorno aislado, típicamente bajo un usuario sin permisos administrativos.

De esta manera, si una vulnerabilidad afecta al código de red,por ejemplo, un error de parsing o de protocolo, el atacante solo obtiene acceso al proceso no privilegiado, sin control sobre el sistema.

Implementación en OpenSSH



En OpenSSH, el modelo de separación se implementa mediante:

- **Fork del proceso principal:** al iniciarse, `sshd` crea un proceso hijo con privilegios reducidos.
- **Comunicación segura entre procesos:** ambos procesos se comunican a través de un canal seguro (por ejemplo, un socket UNIX) para intercambiar información mínima necesaria.
- **Uso de chroot:** el proceso sin privilegios puede ser confinado mediante `chroot()`, limitando aún más su acceso al sistema de archivos.
- **Validación estricta:** todas las acciones sensibles (como la creación de una sesión o la asignación de un terminal) deben ser autorizadas por el proceso privilegiado.

El diseño busca que el código ejecutado con privilegios sea lo más pequeño y simple posible, reduciendo así la probabilidad de errores explotables.

Beneficios de Seguridad

- **Reducción del impacto de vulnerabilidades:** un fallo en el proceso no privilegiado no compromete todo el sistema.
- **Principio de menor privilegio:** cada parte del sistema sólo tiene los permisos necesarios para su función.
- **Mayor resiliencia:** incluso si un atacante obtiene acceso parcial, no puede escalar privilegios fácilmente.

3 Sandboxing

El concepto de **Sandboxing** se refiere a la creación de un **entorno controlado y restringido** en el que las acciones de un proceso o aplicación se limitan de acuerdo con una política de seguridad predefinida. El objetivo principal es **aislar la ejecución** de código potencialmente inseguro o no confiable, de manera que sus efectos no puedan propagarse al resto del sistema.

Este enfoque es ampliamente utilizado en sistemas operativos modernos, navegadores web, entornos de ejecución y servicios en la nube, como una medida de defensa en profundidad frente a vulnerabilidades o comportamientos maliciosos.

Un **sandbox** actúa como una "jaula" (*sandbox*) que impone límites estrictos sobre las operaciones que un proceso puede realizar. El sistema operativo o el entorno de ejecución supervisa y aplica estas restricciones, que pueden incluir:

- Acceso limitado al sistema de archivos.

- Restricciones en la red o comunicación entre procesos.
- Control del uso de memoria y CPU.
- Bloqueo de llamadas al sistema (*syscalls*) potencialmente peligrosas.

De esta forma, incluso si una aplicación es comprometida o ejecuta código malicioso, el daño se mantiene contenido dentro del sandbox, sin afectar la integridad del sistema principal.

3.1 Java Sandbox (deprecado)

La **Java Sandbox** fue uno de los primeros mecanismos de seguridad diseñados para controlar la ejecución de código no confiable dentro del entorno de una máquina virtual. Su objetivo era permitir la ejecución de programas descargados desde la red (principalmente *applets*) sin comprometer la seguridad del sistema del usuario. Aunque este modelo ha sido posteriormente **deprecado**, constituyó un pilar importante en la evolución de las técnicas de aislamiento y control de permisos en tiempo de ejecución.

Applets Java y el modelo de Sandbox

Un **Applet Java** es un programa escrito en Java que puede ser embebido dentro de una página web y ejecutado directamente por el navegador, utilizando la **Java Virtual Machine (JVM)** instalada en el cliente. Por razones de seguridad, estos applets se ejecutaban dentro de la llamada **Sandbox de Java**, un entorno virtual controlado que imponía severas restricciones al código descargado.

Inicialmente, el modelo de Sandbox definía una política de aislamiento muy estricta:

- Los applets no podían acceder a los archivos del sistema local del cliente.
- No se permitía la ejecución o invocación de otros programas del sistema.
- Los applets solo podían establecer comunicación de red con el **host de origen** (es decir, el servidor desde el cual fueron descargados).

De esta forma, la Java Sandbox evitaba que el código remoto pudiera leer, modificar o destruir información local, garantizando que su comportamiento estuviera limitado al entorno virtual provisto por la JVM.

Con el tiempo, se introdujo la posibilidad de que los applets pudieran ejecutar acciones más allá de las restricciones básicas, siempre que provinieran de una fuente considerada confiable. Esto se implementó mediante el uso de **firmas digitales** y la verificación de certificados.

- Si el **applet estaba firmado** por una fuente confiable, se permitía levantar algunas o todas las restricciones impuestas por la sandbox.
- Si el applet **no estaba firmado**, continuaba ejecutándose dentro del entorno altamente restringido original.

Esta extensión permitió desarrollar aplicaciones web más ricas, manteniendo un equilibrio entre funcionalidad y seguridad.

Mecanismo de control de privilegios

Cuando un applet firmado era ejecutado, la **Java Runtime Environment (JRE)** verificaba las políticas de seguridad definidas en el archivo `java.policy`, el cual podía ser configurado mediante la herramienta `policytool`. El comportamiento se determinaba de la siguiente manera:

- Si el applet **no está firmado**, se ejecuta en un entorno altamente restringido (sandbox básica).
- Si el applet **está firmado** y existen reglas específicas en el archivo `java.policy` para su origen (URL), el applet se ejecuta con los privilegios definidos en dicha política.
- Si el applet **está firmado** pero no tiene una política asignada, el JRE verifica si el autor está incluido en la lista de autores confiables:
  - Si lo está, puede otorgársele acceso completo al sistema.
  - Si no lo está, el JRE solicita al usuario una confirmación explícita para permitir o denegar la ejecución.

Deprecación y legado

Con el avance de los navegadores y la aparición de vulnerabilidades asociadas a los plugins, el soporte para **Java Applets** y su modelo de sandbox fue finalmente **deprecado** y eliminado de la mayoría de entornos modernos. Sin embargo, la **Java Sandbox** sentó las bases para mecanismos contemporáneos de aislamiento y control de permisos, influyendo en tecnologías posteriores como los **Security Managers** de la JVM, los **containers** y los sistemas de control de acceso basados en políticas.

4 BSD Security Levels

El sistema operativo **FreeBSD**, al igual que otros derivados del sistema BSD (Berkeley Software Distribution), incorpora un mecanismo de protección adicional denominado **Security Levels**. Este mecanismo permite definir distintos **niveles de seguridad del kernel**, que controlan de manera progresiva qué operaciones están permitidas o prohibidas, incluso para el usuario `root`.

El objetivo principal es **proteger la integridad del sistema** frente a modificaciones accidentales o maliciosas, endureciendo las condiciones de ejecución una vez que el sistema se encuentra en modo multiusuario.

Funcionamiento General

El nivel de seguridad del sistema está representado por una variable de kernel denominada `kern.securelevel`, cuyo valor determina las restricciones aplicadas. A medida que el valor de `securelevel` aumenta, las limitaciones sobre las operaciones del sistema se vuelven más estrictas. Este valor solo puede **incrementarse** mientras el sistema está en ejecución; para disminuirlo, es necesario reiniciar en un modo de menor seguridad (por ejemplo, modo *single-user*).

Niveles de Seguridad BSD

- **-1 – Permanently Insecure Mode:** Modo permanentemente inseguro. El sistema opera siempre como si estuviera en nivel 0. Este modo se utiliza generalmente en entornos de desarrollo o cuando se requiere acceso sin restricciones al hardware y a los dispositivos del sistema.
- **0 – Insecure Mode:** Modo inseguro. En este nivel, las banderas `immutable` y `append-only` pueden ser desactivadas. Todos los dispositivos pueden ser leídos o modificados de acuerdo a sus permisos de archivo. Es el nivel por defecto durante el arranque del sistema o en modo *single-user*.
- **1 – Secure Mode:** Modo seguro. Las banderas `immutable` y `append-only` no pueden ser desactivadas. No se permite la escritura directa a dispositivos montados ni a dispositivos de memoria del kernel como `/dev/mem` o `/dev/kmem`. Además, los módulos del kernel no pueden ser cargados ni descargados dinámicamente. Este modo proporciona una protección efectiva contra modificaciones del sistema en ejecución.
- **2 – Highly Secure Mode:** Modo altamente seguro. Incluye todas las restricciones del modo anterior, agregando medidas adicionales:
  - Los discos solo pueden abrirse para escritura por el proceso de `mount`.
  - No se pueden crear nuevos sistemas de archivos mientras el sistema esté en modo multiusuario.
  - Los cambios a la hora del kernel se restringen a variaciones menores a un segundo, evitando manipulaciones de marca temporal.

  Este modo está orientado a sistemas en producción donde se requiere máxima protección de integridad.
- **3 – Network Secure Mode:** Modo de seguridad de red. Extiende las restricciones del nivel 2, añadiendo la prohibición de modificar la configuración del **firewall** u otros parámetros de red críticos. Está diseñado para entornos donde se prioriza la seguridad sobre la flexibilidad, como servidores expuestos a redes públicas o infraestructuras críticas.

5 Seguridad en Windows: Control de Cuentas de Usuario y Mecanismos de Integridad

A partir de Windows Vista, Microsoft introdujo un modelo de seguridad más granular basado en la separación de privilegios, el control de integridad de objetos y políticas centralizadas de administración. Entre estos mecanismos destacan el **User Account Control (UAC)**, el **Windows Integrity Control**, las cuentas de servicio con privilegios mínimos, y herramientas de control como **AppLocker** y las **Directivas de Grupo (GPO)**.

5.1 Windows User Account Control (UAC)

El **Control de Cuentas de Usuario (User Account Control, UAC)** es un mecanismo de seguridad introducido en Windows Vista que busca limitar el impacto de la ejecución de aplicaciones con privilegios administrativos innecesarios. El UAC previene la ejecución automática de tareas administrativas sin consentimiento explícito, promoviendo el principio de *mínimo privilegio*.

- Cuando un usuario estándar inicia sesión, se crea un **token de sesión** con privilegios básicos.
- Cuando un usuario administrador inicia sesión, el sistema genera **dos tokens de sesión**: uno con todos los privilegios administrativos y otro restringido, con permisos similares a los de un usuario estándar.
- Por defecto, las aplicaciones se ejecutan con el token restringido. Si una aplicación requiere mayores privilegios, el UAC solicita confirmación. Tras la autorización, el proceso se inicia con el token sin restricciones.

Acciones que disparan la ventana del UAC

- Ejecutar una aplicación como administrador.
- Modificar configuraciones que afecten al sistema completo o a archivos en `%SystemRoot%` o `%ProgramFiles%`.
- Instalar o desinstalar aplicaciones o controladores de dispositivos.
- Instalar controles ActiveX.
- Cambiar la configuración del firewall o del propio UAC.
- Configurar Windows Update o el control parental.
- Administrar cuentas de usuario.
- Ejecutar el *Task Scheduler*, restaurar backups o desfragmentar discos.
- Ver o modificar archivos de otros usuarios.

5.2 Windows Integrity Control

Windows Integrity Control (WIC), inicialmente conocido como **Mandatory Integrity Control (MIC)**, fue introducido en Windows Vista. Se basa en el modelo **Biba** de control de integridad, que busca impedir que procesos de menor integridad modifiquen objetos de mayor integridad.

- Define seis niveles de integridad:
  - **Trusted Installer**
  - **System** (procesos del sistema operativo)
  - **High** (administradores)
  - **Medium** (usuarios estándar)
  - **Low** (archivos temporales de Internet)
  - **Untrusted**
- Archivos, carpetas, usuarios y procesos poseen etiquetas de integridad.
- El nivel *Medium* es el valor predeterminado para usuarios estándar.
- Un usuario no puede asignar a un objeto un nivel superior al suyo propio.

5.3 Servicios y mínimos privilegios en Windows 2003

Windows 2003 introdujo un modelo más seguro para la ejecución de servicios, con diferentes cuentas de sistema diseñadas para limitar el alcance de privilegios:

- **Local Service:** privilegios mínimos en el equipo local; acceso a la red mediante sesión nula y credenciales anónimas.
- **Network Service:** privilegios mínimos locales; acceso a la red con credenciales de la cuenta de equipo.
- **Unique User Account:** usada cuando no es viable ejecutar el servicio bajo las cuentas anteriores.
- **Local System:** amplios privilegios en el sistema local y acceso a recursos de red mediante credenciales del equipo.
- **Local Administrator Account:** debe evitarse salvo necesidad operativa.
- **Domain Administrator Account:** su uso para servicios representa un riesgo crítico de seguridad.

5.4 Mejoras en Windows 2008

Service	Previous Context	New Context
COM+ Event System	SYSTEM	LOCAL SERVICE
Windows Security	SYSTEM	LOCAL SERVICE
Windows Event Log	SYSTEM	LOCAL SERVICE
Windows Audio	SYSTEM	LOCAL SERVICE
Workstation Service	SYSTEM	LOCAL SERVICE
Windows Image Acquisition	SYSTEM	LOCAL SERVICE
Windows Time	SYSTEM	LOCAL SERVICE
DHCP Client	SYSTEM	LOCAL SERVICE
Telephony	SYSTEM	NETWORK SERVICE
Cryptographic Services	SYSTEM	NETWORK SERVICE
Policy Agent	SYSTEM	NETWORK SERVICE
Terminal Services	SYSTEM	NETWORK SERVICE

Table 12-1 Vista Services that Have Now Run Under Lower-privileged Accounts

Windows Server 2008 reforzó la seguridad mediante:

- Integración del modelo UAC en el núcleo del sistema.
- Mejoras en la administración de roles y servicios.
- Nuevas políticas de auditoría y aislamiento de servicios.

5.5 Windows AppLocker

AppLocker permite establecer reglas para controlar qué software puede ejecutarse en un entorno Windows, basándose en firmas digitales y atributos de archivo.

- Controla archivos ejecutables (`.exe`, `.com`), scripts (`.js`, `.ps1`, `.vbs`, `.cmd`, `.bat`), instaladores (`.msi`, `.msp`) y bibliotecas dinámicas (`.dll`, `.ocx`).
- Permite crear reglas basadas en el **editor**, **producto**, **nombre de archivo** o **versión**.
- Las reglas se asignan a usuarios o grupos de seguridad específicos.
- Se pueden crear excepciones (por ejemplo, permitir todo excepto `Regedit.exe`).
- Soporta un **modo de auditoría** para evaluar el impacto antes de aplicar políticas.
- Su gestión puede automatizarse mediante **PowerShell**.

5.6 Windows 2012 y 2016

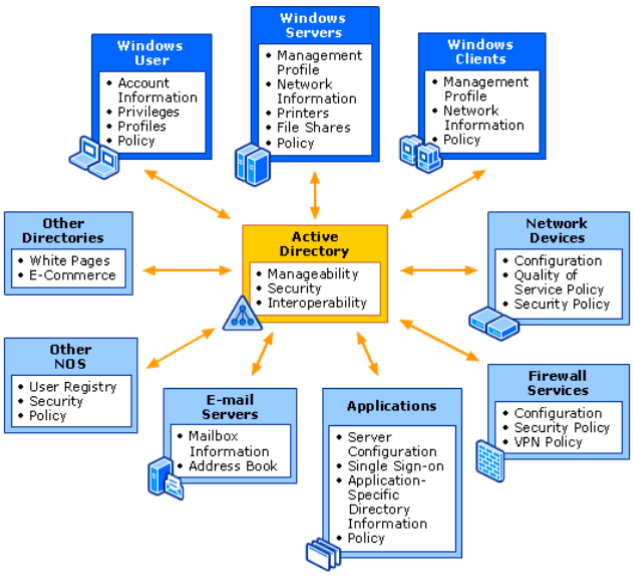
Windows Server 2012

- Mejoras en **BitLocker** y soporte para **UEFI Secure Boot**.
- Introducción de **Dynamic Access Control** para gestión de permisos basada en atributos.

Windows Server 2016

- **Credential Guard:** protege credenciales mediante virtualización.
- **Device Guard:** permite ejecutar únicamente código firmado y verificado.
- **Host Guardian Service** y **Shielded Virtual Machines:** refuerzan la seguridad en entornos virtualizados.

5.7 Active Directory



Active Directory (AD) es el servicio de directorio de Microsoft diseñado para centralizar la administración y organización de recursos dentro de un dominio. Su propósito principal es proporcionar un modelo coherente para gestionar usuarios, equipos, aplicaciones y servicios de red, garantizando al mismo tiempo la seguridad y la interoperabilidad en entornos heterogéneos.

Los tres pilares fundamentales de Active Directory son:

- **Manageability (Administrabilidad):** AD permite a los administradores centralizar la gestión de usuarios, grupos, políticas, equipos y recursos de red, simplificando tareas como la asignación de permisos, la creación de cuentas y la delegación de autoridad.
- **Security (Seguridad):** AD integra mecanismos de autenticación y autorización robustos, como Kerberos y LDAP con seguridad, control de acceso basado en roles (RBAC), políticas de contraseñas, y auditoría de eventos, para proteger los recursos de la organización.
- **Interoperability (Interoperabilidad):** AD soporta la integración con otros sistemas operativos, directorios y servicios de red, permitiendo coexistir y federar recursos con servicios como LDAP, UNIX/Linux NIS/NIS+, otros servidores de directorio y aplicaciones empresariales.

Componentes gestionados por Active Directory

Active Directory permite organizar y administrar de forma centralizada una amplia gama de recursos dentro de la red corporativa:

- **Usuarios, servidores y clientes Windows:** AD centraliza cuentas de usuario y grupos, aplicando políticas de seguridad y configuraciones mediante *Group Policy Objects (GPO)*. También gestiona equipos Windows dentro del dominio, asegurando coherencia en configuraciones y actualizaciones.
- **Network Devices:** Algunos dispositivos de red como switches, routers y puntos de acceso pueden integrarse con AD para autenticación centralizada mediante protocolos como RADIUS o 802.1X.
- **Firewall Services:** AD permite aplicar políticas de acceso centralizadas a firewalls corporativos, gestionando reglas y permisos según roles y grupos definidos en el directorio.
- **Aplicaciones empresariales:** Servidores de aplicaciones pueden usar AD para autenticar usuarios y definir permisos de acceso a recursos críticos.
- **Servidores de correo electrónico:** Servicios como Microsoft Exchange se integran estrechamente con AD para la gestión de buzones, listas de distribución, roles administrativos y seguridad de correo.
- **Otros sistemas operativos y servicios de directorio:** AD puede coexistir e interoperar con sistemas UNIX/Linux, otros servidores de directorio como LDAP, y servicios de autenticación federada, garantizando un modelo de identidad unificado.

Beneficios de Active Directory

- **Administración centralizada:** La gestión de usuarios, equipos, políticas y recursos se realiza desde un único punto de control.
- **Seguridad reforzada:** Autenticación Kerberos, control de acceso granular, auditoría de eventos y políticas de contraseñas permiten proteger la infraestructura corporativa.
- **Escalabilidad y flexibilidad:** AD soporta dominios múltiples, bosques y relaciones de confianza, facilitando la expansión de la red sin comprometer la integridad de los datos.
- **Interoperabilidad:** La capacidad de integrarse con otros sistemas operativos y directorios permite gestionar redes heterogéneas desde un único directorio central.
- **Eficiencia operativa:** La implementación de *Group Policy*, scripts de inicio de sesión y plantillas de seguridad reduce la carga administrativa y asegura consistencia en toda la organización.

5.8 Políticas de Grupo y Plantillas de Seguridad en Windows

En entornos corporativos, mantener configuraciones de seguridad y operativas consistentes en todos los equipos y usuarios es fundamental. Windows ofrece para ello las **Directivas de Grupo (Group Policy Objects, GPO)**, que permiten a los administradores centralizar la configuración y aplicación de políticas en dominios Active Directory. Complementariamente, las **plantillas de seguridad (Security Templates)** permiten definir configuraciones prediseñadas que pueden aplicarse de manera uniforme en toda la organización.

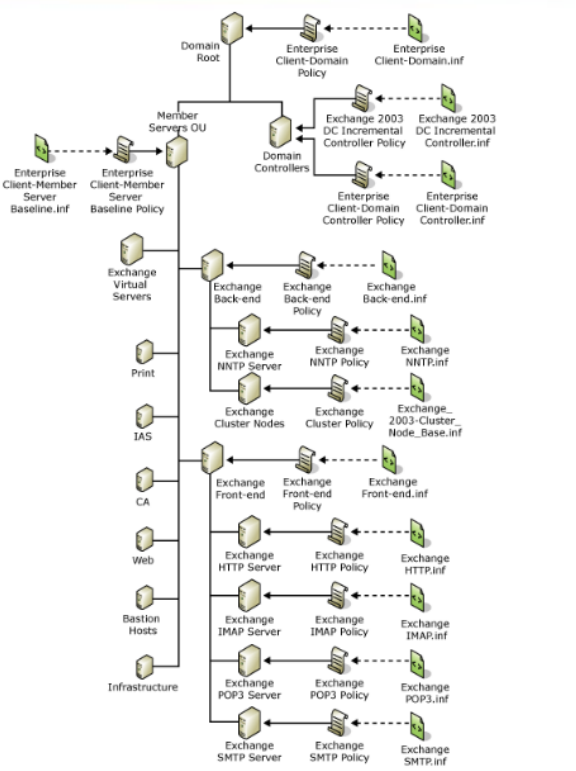
Funciones y ventajas de las Directivas de Grupo

- Permiten aplicar configuraciones coherentes a grupos de usuarios y equipos, garantizando uniformidad en la administración del sistema.
- Facilitan el cumplimiento de políticas corporativas escritas. Por ejemplo, una organización puede requerir que todos los equipos de un departamento muestren un mensaje de advertencia al iniciar sesión, informando sobre la supervisión de seguridad en dicho departamento. Las GPO permiten configurar, implementar y administrar este tipo de mensajes de manera centralizada.
- Posibilitan la gestión centralizada de parámetros de seguridad, aplicaciones, scripts de inicio de sesión, configuraciones de red y otras políticas administrativas.

Plantillas de Seguridad (Security Templates)

- Las plantillas de seguridad permiten empaquetar un conjunto de configuraciones de seguridad en un solo archivo, listo para ser aplicado a múltiples equipos cliente.
- Este enfoque asegura que todos los equipos tengan una configuración coherente, reduciendo errores de configuración manual y garantizando el cumplimiento de normas corporativas.
- Al centralizar la gestión de estas plantillas, los administradores pueden revisar, actualizar y mejorar continuamente las políticas de seguridad a lo largo del tiempo, adaptándose a nuevos requisitos o vulnerabilidades detectadas.
- La combinación de GPO con plantillas de seguridad permite una implementación escalable, auditable y repetible de configuraciones críticas para la seguridad del entorno Windows.

Unidades Organizativas (OUs) y Políticas



Dentro de **Active Directory**, las **Unidades Organizativas (Organizational Units, OUs)** constituyen contenedores lógicos que permiten organizar usuarios, grupos y equipos de manera jerárquica y funcional. Las OUs no solo facilitan la administración estructurada de los recursos, sino que también son esenciales para aplicar políticas específicas de forma granular.

- **Asignación de políticas:** Mediante las OUs, los administradores pueden vincular *Group Policy Objects (GPOs)* a un subconjunto específico de usuarios o equipos, aplicando configuraciones de seguridad, restricciones de software y ajustes administrativos según necesidades organizativas.
- **Delegación de autoridad:** Las OUs permiten delegar tareas administrativas de manera segura. Por ejemplo, un administrador de departamento puede gestionar cuentas de usuarios dentro de su OU sin afectar a otros segmentos de la organización.
- **Organización jerárquica:** Se pueden anidar OUs para reflejar la estructura organizativa de la empresa, lo que permite aplicar políticas de forma acumulativa y coherente.
- **Seguridad y consistencia:** La vinculación de políticas a OUs asegura que configuraciones críticas de seguridad se apliquen automáticamente, evitando la dispersión de configuraciones y la intervención manual.

5.9 Microsoft LAPS (Local Administrator Password Solution)

Microsoft LAPS es una herramienta de seguridad diseñada para la gestión automatizada de las contraseñas de cuentas locales de administrador en equipos dentro de un dominio. Su objetivo es reducir riesgos asociados al uso de contraseñas compartidas o estáticas en entornos corporativos.

- **Generación automática de contraseñas:** LAPS genera contraseñas únicas y complejas para cada equipo, evitando que todas las máquinas compartan la misma credencial administrativa.
- **Almacenamiento seguro:** Las contraseñas se guardan de forma cifrada en Active Directory, accesibles solo para usuarios o grupos con permisos específicos.
- **Rotación periódica:** LAPS permite configurar la caducidad de contraseñas y su renovación automática, reduciendo el riesgo de exposición prolongada.

- **Auditoría y control:** Cada acceso a la contraseña administrada puede ser registrado, permitiendo un seguimiento de uso y mejorando la trazabilidad en auditorías de seguridad.
- **Integración con OUs y políticas:** LAPS se despliega fácilmente mediante GPOs, aplicando configuraciones de gestión de contraseñas de manera consistente según la estructura de OUs del dominio.

6 Linux Capabilities y Sudo

En sistemas operativos tipo Unix la gestión de privilegios es crítica para la seguridad del sistema. Tradicionalmente, el usuario **root** posee todos los privilegios, lo que plantea riesgos de seguridad si un proceso o usuario malintencionado obtiene acceso a dicha cuenta. Para mitigar estos riesgos, Linux implementa mecanismos como **Capabilities** y la utilidad **sudo**, que permiten otorgar privilegios de forma granular y controlada.

Linux Capabilities

Linux Capabilities son un mecanismo que permite dividir los privilegios tradicionales del usuario root en unidades más pequeñas, asignables a procesos o ejecutables específicos. De esta manera, un programa puede ejecutar acciones privilegiadas sin necesidad de correr como root, reduciendo la superficie de ataque del sistema.

- Permite otorgar permisos privilegiados de manera **restringida y controlada**.
- Ejemplo de uso:

  ```
  $ getcap /usr/bin/dumpcap
  /usr/bin/dumpcap = cap_net_admin,cap_net_raw+eip
  ```

  En este ejemplo, el binario `/usr/bin/dumpcap` tiene capacidades para administrar la red (`cap_net_admin`) y acceso crudo a la red (`cap_net_raw`), con las flags `eip` que indican efectos sobre herencia y permisos.
- La utilidad `man capabilities` permite consultar la lista completa de capacidades disponibles y su funcionalidad.
- Este enfoque refuerza la seguridad mediante el principio de *mínimo privilegio*, otorgando a cada proceso solo los privilegios estrictamente necesarios para su operación.

Sudo

Sudo (del inglés *superuser do* o *substitute user do*) es una herramienta que permite a los usuarios ejecutar programas con los privilegios de otro usuario, normalmente root, sin necesidad de compartir la contraseña de este último.

- Permite otorgar privilegios temporales y controlados a usuarios específicos.
- Su configuración se realiza mediante el archivo `/etc/sudoers`, donde se pueden definir qué comandos puede ejecutar cada usuario y bajo qué condiciones.
- Ejemplo de uso:

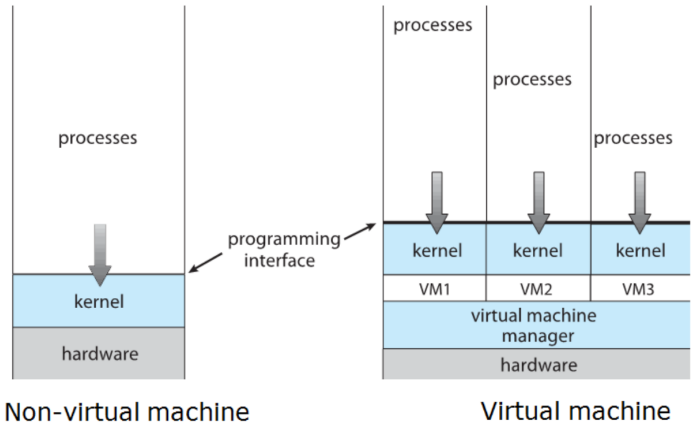
  ```
  $ sudo apt update
  ```

  Aquí, el usuario ejecuta el comando `apt update` con privilegios de superusuario, sin necesidad de iniciar sesión como root.
- Sudo ofrece trazabilidad: todos los comandos ejecutados quedan registrados en los logs del sistema, lo que permite auditoría y control de acciones administrativas.
- Combinado con Linux Capabilities, sudo permite un modelo de seguridad flexible y granular, donde los usuarios pueden realizar tareas administrativas necesarias sin comprometer la integridad del sistema.

7 Máquinas Virtuales y Contenedores

En el ámbito de la infraestructura de software moderna, la **virtualización** y la **contenedorización** son dos enfoques fundamentales para el despliegue y la administración de aplicaciones. Ambas tecnologías buscan aislar procesos y optimizar el uso de los recursos del sistema, pero difieren en su arquitectura, nivel de abstracción y casos de uso.

La virtualización se basa en la creación de **máquinas virtuales (Virtual Machines, VMs)** que simulan hardware completo, mientras que los **contenedores** operan a nivel del sistema operativo, compartiendo el mismo kernel pero aislando los entornos de ejecución de las aplicaciones.



Non-Virtual Machine (Ejecución directa sobre hardware)

En un entorno **non-virtual machine**, el sistema operativo se ejecuta directamente sobre el **hardware físico** del equipo. En este modelo tradicional:

- El sistema operativo controla directamente la CPU, la memoria y los dispositivos de E/S.
- Las aplicaciones se ejecutan sobre el sistema operativo sin ningún nivel intermedio de virtualización.
- La eficiencia es máxima, pero no existe aislamiento entre sistemas o versiones distintas de entornos.

Este modelo es adecuado cuando se requiere **rendimiento máximo** y no se necesita ejecutar múltiples sistemas operativos en paralelo, pero limita la escalabilidad y la portabilidad.

7.1 Virtual Machine

Una **máquina virtual (VM)** es un entorno de ejecución aislado que **emula un sistema completo**, incluyendo CPU, memoria, almacenamiento y dispositivos. Cada VM ejecuta su propio sistema operativo invitado (*guest OS*) sobre un sistema operativo anfitrión (*host OS*).

El componente clave que permite esto es el **hipervisor**.

7.1.1 Hypervisor

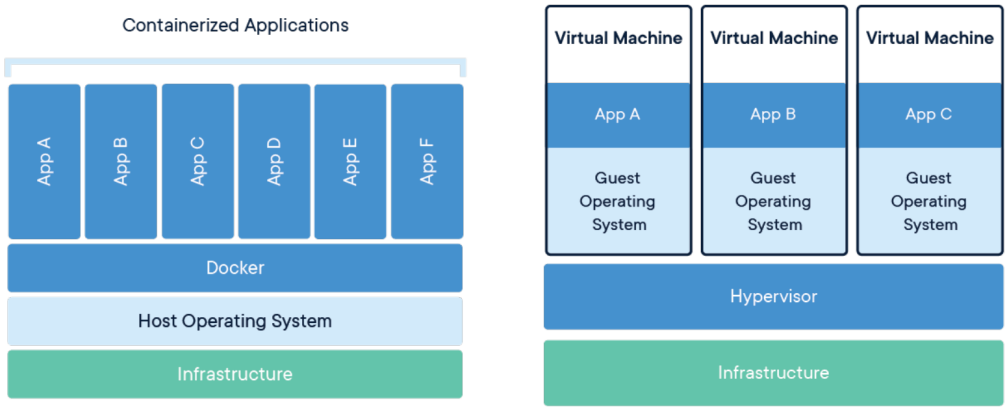
El **hipervisor** (o *Virtual Machine Monitor, VMM*) es la capa de software responsable de abstraer y gestionar los recursos físicos, permitiendo que múltiples máquinas virtuales coexistan en el mismo hardware.

Existen dos tipos principales de hipervisores:

- **Tipo 1 (bare-metal):** se ejecuta directamente sobre el hardware físico. Ejemplos: VMware ESXi, Microsoft Hyper-V, Xen.
- **Tipo 2 (hosted):** se ejecuta sobre un sistema operativo anfitrión. Ejemplos: VirtualBox, VMware Workstation.

Cada VM posee su propio kernel, sistema de archivos y pila de red. Esto proporciona un **aislamiento fuerte** y la capacidad de ejecutar sistemas operativos completamente diferentes en un mismo servidor. Sin embargo, la virtualización tradicional presenta una **sobrecarga significativa** de recursos, ya que cada VM requiere un sistema operativo completo y un entorno de hardware emulado.

7.2 Contenedores



Los **contenedores** son una evolución de la virtualización orientada a la eficiencia. En lugar de emular hardware, los contenedores utilizan mecanismos del sistema operativo —como **namespaces**, **cgroups** y **capabilities** en Linux— para proporcionar aislamiento entre aplicaciones.

Cada contenedor incluye:

- Su propia imagen del sistema de archivos.
- Dependencias y librerías necesarias para la aplicación.
- Un entorno aislado de otros contenedores.

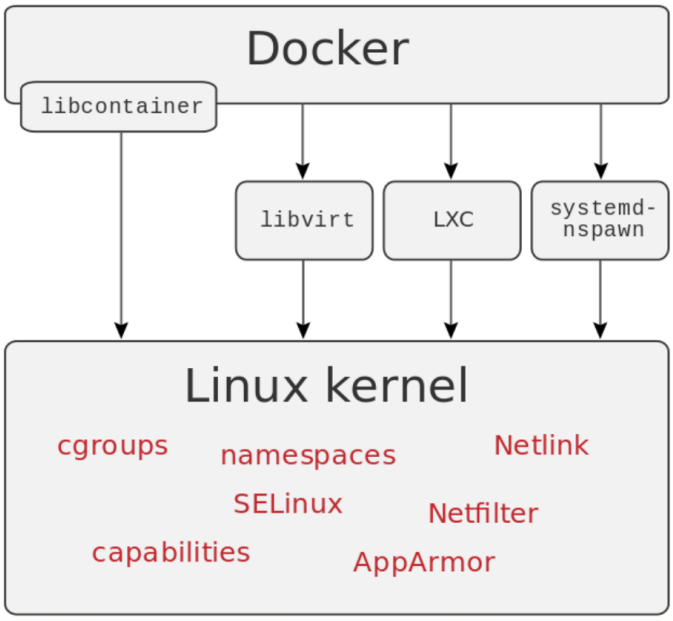
A diferencia de las VMs, **todos los contenedores comparten el mismo kernel del host**, lo que los hace **más ligeros** y **rápidos de iniciar**. No se requiere un hipervisor, sino un **motor de contenedores (container engine)**, como **Docker** o **Podman**, que gestiona la creación, ejecución y comunicación entre contenedores.

Mecanismos de aislamiento en Linux

Los contenedores se apoyan en varias funcionalidades del kernel de Linux para lograr aislamiento, control de recursos y seguridad:

- **cgroups (Control Groups):** permiten limitar y monitorizar el uso de recursos del sistema, como CPU, memoria, E/S o red, asignando cuotas y prioridades a procesos o grupos de procesos.
- **namespaces:** proporcionan aislamiento de vistas del sistema. Cada contenedor puede tener su propio espacio de procesos, red, sistema de archivos, usuarios y montajes, sin interferir con el sistema host.
- **netlink:** es una interfaz de comunicación entre el espacio de usuario y el kernel utilizada para gestionar configuraciones de red, como interfaces virtuales, rutas o namespaces de red.
- **netfilter:** es el marco del kernel de Linux para inspeccionar y manipular paquetes de red. Se utiliza junto con `iptables` o `nftables` para aplicar reglas de filtrado, NAT o control de tráfico dentro de contenedores.
- **AppArmor:** es un sistema de control de acceso obligatorio (MAC) que restringe las acciones que los procesos pueden realizar, definiendo perfiles de seguridad por aplicación o contenedor.
- **capabilities:** dividen los privilegios del usuario root en permisos individuales más granulares, permitiendo otorgar solo las capacidades necesarias a un contenedor sin conceder acceso total al sistema.

7.3 Docker y el ecosistema de contenedores



Docker es la tecnología más popular para la gestión de contenedores en sistemas Linux. Define un formato estándar para construir, empaquetar y distribuir aplicaciones portables junto con todas sus dependencias, garantizando consistencia en su ejecución sin importar el entorno subyacente.

El funcionamiento de Docker se basa en tres conceptos principales:

1. **Imágenes (Images):** plantillas inmutables que contienen el entorno de ejecución de la aplicación, el sistema de archivos base y sus dependencias.
2. **Contenedores (Containers):** instancias en ejecución de una imagen. Cada contenedor es un proceso aislado que utiliza los mecanismos del kernel (namespaces, cgroups, capabilities) para mantener independencia del sistema host.
3. **Docker Engine:** el demonio responsable de crear, ejecutar y gestionar contenedores, comunicándose con el kernel de Linux mediante una API.

7.3.1 Arquitectura interna de Docker

Internamente, Docker utiliza **libcontainer**, una biblioteca desarrollada originalmente por el propio proyecto Docker (y ahora parte de `runc`) que provee una interfaz directa con las primitivas del kernel de Linux: `cgroups`, `namespaces`, `AppArmor`, `seccomp`, entre otras. Antes de la aparición de `libcontainer`, Docker dependía de **LXC (Linux Containers)** como backend por defecto. LXC es una tecnología de contenedores a bajo nivel que ofrece herramientas y APIs para crear y administrar contenedores sin requerir hipervisores, utilizando directamente los mecanismos del kernel.

libcontainer reemplazó a LXC en Docker como capa de abstracción más flexible y segura, permitiendo control granular sobre la configuración de cada contenedor. Con el tiempo, esta biblioteca fue estandarizada en el proyecto **Open Container Initiative (OCI)**, que define especificaciones comunes para el formato de imágenes y el runtime de contenedores, representado por la herramienta `runc`.

Sistema de archivos en capas (Layered Filesystem)

Un componente esencial del diseño de Docker es el uso de un **sistema de archivos en capas** (*Layered Filesystem*). Cada imagen de Docker está compuesta por múltiples capas superpuestas, donde cada capa representa un conjunto de cambios sobre la anterior (por ejemplo, instalación de paquetes, copiado de archivos, configuraciones, etc.).

Estas capas son gestionadas mediante controladores de almacenamiento como `OverlayFS`, `AUFS`, `btrfs` o `ZFS`, que permiten compartir partes comunes entre contenedores de manera eficiente.

- Las capas inferiores son **de solo lectura** y se comparten entre diferentes contenedores.
- La capa superior, asociada al contenedor en ejecución, es **de lectura y escritura**, donde se registran los cambios locales.
- Este enfoque reduce significativamente el uso de espacio en disco y acelera la creación de nuevas imágenes.

Gracias al **Layered FS**, Docker puede construir imágenes de manera incremental, reutilizando capas previas y evitando duplicación de datos. Este modelo también facilita la distribución de imágenes a través de repositorios como **Docker Hub**, ya que solo las capas nuevas deben ser transferidas.

Integración con otras tecnologías de virtualización

Aunque Docker es la herramienta más extendida en el ámbito de la contenedorización, existen otros frameworks y herramientas con propósitos similares o complementarios:

- **libvirt:** una API y conjunto de herramientas para administrar plataformas de virtualización. Aunque se asocia principalmente a máquinas virtuales (como KVM o QEMU), también ofrece soporte para contenedores mediante `libvirt-lxc`, integrando gestión unificada de VMs y contenedores.
- **LXC (Linux Containers):** una tecnología nativa del kernel de Linux que proporciona una interfaz más directa y manual para crear contenedores. Permite definir parámetros de red, recursos y dispositivos con mayor control, siendo ideal para entornos de sistema o virtualización ligera tradicional.
- **systemd-nspawn:** herramienta incluida en `systemd` que permite ejecutar comandos o sistemas completos en entornos aislados, similar a un contenedor. A diferencia de Docker, su foco es la administración de sistemas y entornos reproducibles, más que el despliegue de aplicaciones. Se utiliza habitualmente para pruebas, construcción de imágenes o entornos tipo "chroot" extendidos.

Ecosistema

Docker no solo define un formato de contenedores, sino también un ecosistema completo para su gestión. Incluye:

- **Docker Hub:** repositorio público de imágenes para compartir y distribuir aplicaciones.
- **Docker Compose:** herramienta para definir entornos multi-contenedor mediante archivos YAML.
- **Docker Swarm y Kubernetes:** plataformas de orquestación que permiten escalar, distribuir y administrar múltiples contenedores en clústeres.

7.4 Seguridad en Máquinas Virtuales y Contenedores

La **seguridad en entornos virtualizados y contenedorizados** constituye un campo complejo y multidimensional dentro de la **Seguridad de la Información**. Cada capa —desde el hardware hasta las aplicaciones— introduce posibles vectores de ataque y requiere mecanismos específicos de protección.

Las máquinas virtuales (VMs) y los contenedores comparten objetivos de aislamiento, pero difieren en su nivel de abstracción: las VMs operan a nivel de hardware virtualizado mediante un *hypervisor*, mientras que los contenedores se apoyan en el kernel del sistema operativo. En consecuencia, los riesgos y estrategias de mitigación son distintos para cada caso.

Superficie de ataque y aislamiento

El principal desafío en entornos virtualizados es evitar que un atacante que compromete una VM o un contenedor pueda **escapar hacia el host** o hacia otras instancias. Este tipo de ataque, conocido como *VM escape* o *container breakout*, puede permitir la ejecución de código arbitrario en el sistema físico subyacente.

- En las **máquinas virtuales**, la protección depende del hipervisor. Se utilizan técnicas de separación de memoria, control de dispositivos de E/S y aislamiento de CPU para evitar que una VM acceda a recursos ajenos.
- En los **contenedores**, el aislamiento se logra mediante mecanismos del kernel Linux como `namespaces`, `cgroups`, `AppArmor`, `SELinux` y `seccomp`. Estos limitan el alcance del proceso dentro de su entorno de ejecución y restringen las llamadas al sistema.

La configuración incorrecta o la concesión excesiva de privilegios (por ejemplo, ejecutar contenedores con la opción `--privileged`) puede anular este aislamiento, exponiendo el host a ataques.

7.4.1 Secure Supply Chain

La **cadena de suministro de software** es otro componente crítico. En entornos donde se despliegan imágenes y contenedores provenientes de múltiples fuentes, es esencial garantizar la autenticidad e integridad del software.

- Las imágenes de contenedores deben ser firmadas digitalmente y verificadas antes del despliegue.
- Se recomienda el uso de herramientas como **Docker Content Trust**, **Cosign** o **Notary** para validar la procedencia.
- Los procesos de **DevSecOps** integran análisis estáticos, escaneo de vulnerabilidades y políticas de aprobación continua a lo largo del ciclo de vida del software.

Estas prácticas conforman el concepto de **Secure Software Supply Chain**, orientado a reducir riesgos derivados de código malicioso, dependencias comprometidas o vulnerabilidades no corregidas en imágenes base.

7.4.2 Software Defined Networks (SDN)

En infraestructuras modernas, tanto las VMs como los contenedores suelen interconectarse mediante redes virtuales. Las **Software Defined Networks (SDN)** permiten gestionar y controlar dinámicamente la red mediante software, separando el plano de control del plano de datos.

Desde el punto de vista de la seguridad:

- SDN habilita políticas centralizadas de segmentación, detección de intrusiones y microsegmentación entre contenedores o VMs.
- Facilita la implementación de **firewalls distribuidos** y el aislamiento de tráfico a nivel de capa 2/3, reduciendo la propagación lateral de ataques.
- Permite instrumentar medidas de *zero trust*, donde cada conexión se autentica y verifica independientemente.

7.4.3 Trusted Execution Environment (TEE)

Un componente clave en la evolución de la seguridad de la virtualización es el **Trusted Execution Environment (TEE)**. Se trata de un área protegida dentro de la CPU que permite ejecutar código y almacenar datos confidenciales de forma aislada, incluso frente al sistema operativo o al hipervisor.

- El TEE crea un entorno de ejecución seguro donde los procesos pueden operar sobre información sensible (como claves criptográficas o datos biométricos) sin riesgo de exposición.
- Está diseñado para resistir tanto ataques de software como ataques físicos, protegiendo la confidencialidad y la integridad de los datos.

Existen diferentes implementaciones de TEE según la arquitectura del procesador:

- **Intel SGX (Software Guard Extensions):** proporciona enclaves seguros dentro del procesador donde se ejecuta código protegido y cifrado, inaccesible incluso para el sistema operativo.
- **AMD SEV (Secure Encrypted Virtualization):** implementa cifrado transparente de la memoria de las VMs, garantizando que el hipervisor no pueda leer su contenido.
- **ARM TrustZone:** divide el sistema en dos mundos —seguro y no seguro—, permitiendo ejecutar operaciones críticas en el entorno protegido, como la gestión de claves o la autenticación biométrica.

8 Security-Enhanced Linux (SELinux)

Security Enhanced Linux (SELinux) es una extensión del núcleo de Linux diseñada para imponer políticas de control de acceso estrictas bajo el paradigma de *Mandatory Access Control* (MAC). Fue originalmente difundida por la Agencia de Seguridad Nacional (NSA) de los Estados Unidos en el año 2000, con la colaboración de NAI Labs, Secure Computing Corporation y MITRE Corporation. El código fuente fue liberado a la comunidad *Open Source* y posteriormente integrado de manera oficial al núcleo Linux a partir de la versión 2.6, convirtiéndose en una parte fundamental del marco de seguridad del sistema operativo.

A diferencia de los mecanismos tradicionales de seguridad en Linux, SELinux proporciona una capa adicional que regula cómo los procesos interactúan con los objetos del sistema (archivos, dispositivos, sockets, etc.), incluso cuando los permisos estándar del sistema lo permitirían. Su objetivo es limitar el daño potencial ante una intrusión o explotación de vulnerabilidades, aplicando el principio del *mínimo privilegio* y confinando cada proceso dentro de un dominio controlado.

8.1 Modelos de Control de Acceso

SELinux se basa en el reemplazo del modelo de control de acceso tradicional de Unix, denominado **DAC (Discretionary Access Control)**, por un modelo más estricto conocido como **MAC (Mandatory Access Control)**.

Control de Acceso Discrecional (DAC)

El modelo DAC es el utilizado por los sistemas Unix tradicionales. En este enfoque:

- Las decisiones de acceso se basan en la identidad del usuario y la propiedad de los objetos.
- Cada usuario tiene control total sobre los permisos de sus archivos y procesos.
- Los procesos heredan los privilegios del usuario que los ejecuta.
- No existe un control del flujo de información entre usuarios si estos tienen los permisos adecuados sobre los mismos objetos.

Si bien DAC es flexible, esta flexibilidad representa una debilidad: un proceso comprometido hereda los privilegios del usuario, pudiendo acceder o modificar recursos que deberían estar fuera de su alcance.

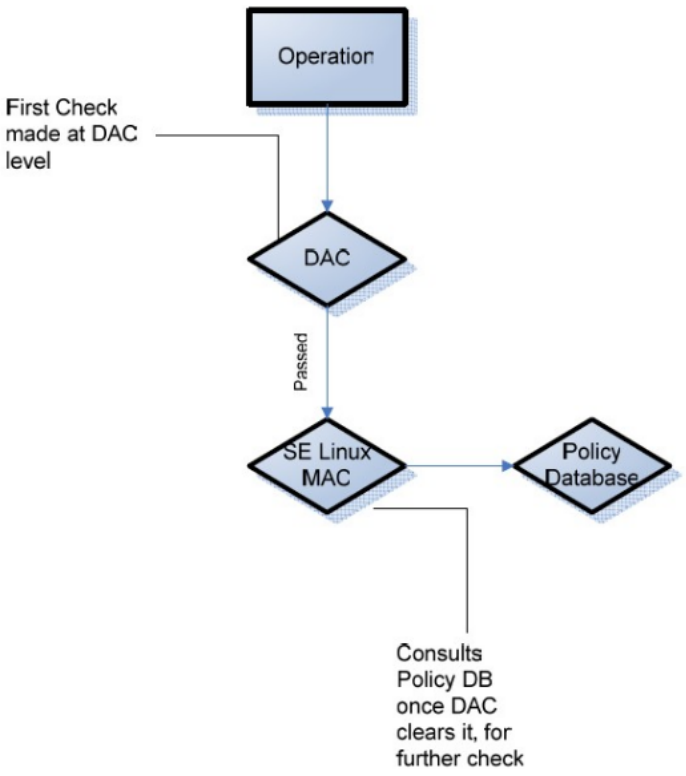
Control de Acceso Obligatorio (MAC)

El modelo MAC, utilizado por SELinux, impone un conjunto de políticas globales definidas por el administrador y forzadas por el sistema operativo. Este modelo se caracteriza por:

- Reglas de acceso explícitas que no pueden ser alteradas por los usuarios finales.
- Separación de dominios y recursos mediante políticas de aislamiento.
- Reglas de integridad que evitan modificaciones no autorizadas sobre datos o aplicaciones.
- Asignación de roles y tipos específicos que limitan el alcance de las operaciones de cada proceso.

De esta forma, aunque un proceso o usuario con privilegios sea comprometido, las políticas MAC previenen que dicho proceso acceda a recursos fuera de su dominio de seguridad.

8.2 Funcionamiento Interno de SELinux



SELinux introduce nuevos conceptos para describir la relación entre procesos, objetos y permisos dentro del sistema:

- **Sujetos**: los procesos o entidades que realizan acciones.
- **Objetos**: los recursos sobre los que se realizan las operaciones (archivos, sockets, directorios, dispositivos, etc.).
- **Operaciones**: las acciones que los sujetos intentan realizar sobre los objetos.

Cada objeto puede clasificarse dentro de una *clase de objetos* (*object class*), lo que permite definir reglas de acceso específicas por tipo de recurso.

El sistema de control de acceso tradicional (DAC) sigue teniendo prioridad inicial: si el acceso es denegado por los permisos estándar de Unix, SELinux no interviene. Sin embargo, si DAC lo permite, entonces SELinux aplica su propia política MAC para determinar si la acción es válida bajo el contexto de seguridad configurado.

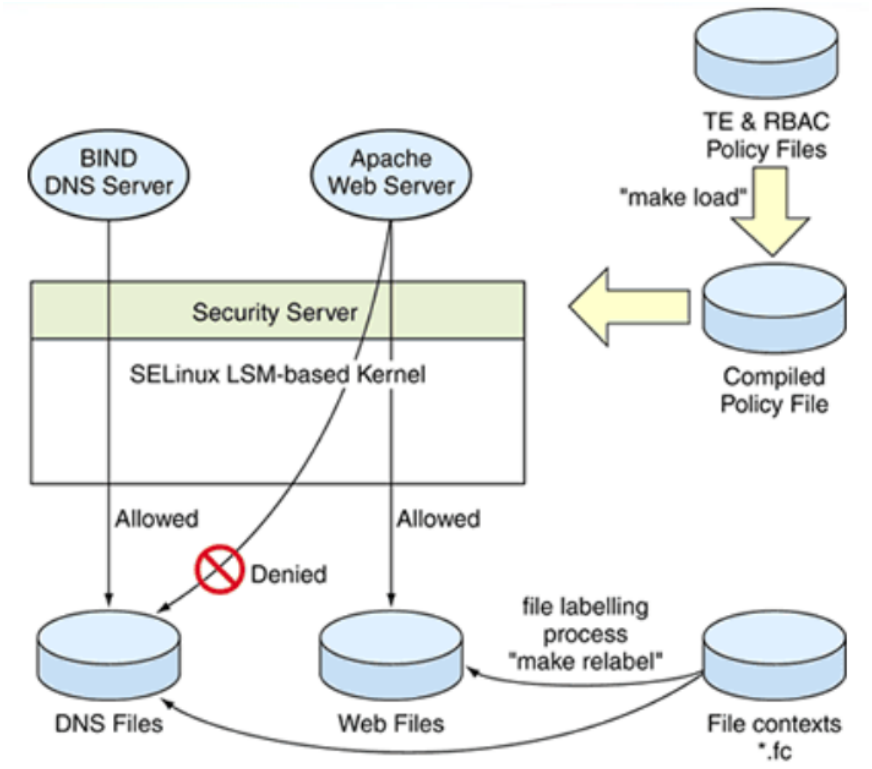
8.2.1 Implementación en el Núcleo

SELinux se implementa como un **Linux Security Module (LSM)**, una infraestructura que permite insertar mecanismos de control de seguridad directamente en el kernel. A través de *hooks* distribuidos en puntos críticos del sistema (llamadas al sistema, acceso a archivos, comunicaciones IPC, etc.), cada operación realizada por un proceso es interceptada y validada contra las políticas de seguridad de SELinux.

De esta manera, el kernel consulta a SELinux antes de permitir la ejecución de una operación. Esto garantiza que ninguna acción fuera de las políticas definidas pueda completarse, incluso si el proceso tiene permisos convencionales suficientes.

Para ejecutar un programa bajo un entorno SELinux, el administrador debe comprender qué archivos, subprocesos y recursos serán afectados, y cómo las políticas los regulan.

8.2.2 Dominios y Confinamiento



SELinux utiliza el concepto de **dominios** para aislar las aplicaciones. Cada proceso se ejecuta dentro de un dominio específico, el cual determina las acciones permitidas y los recursos accesibles. Este confinamiento asegura que una aplicación comprometida no pueda afectar a otras ni acceder a información fuera de su contexto.

Por ejemplo, un proceso del navegador web podría ejecutarse en el dominio `firefox_t`, mientras que los archivos del usuario están etiquetados con el tipo `user_home_t`. Las políticas de SELinux definirán si el proceso `firefox_t` puede leer, escribir o ejecutar objetos del tipo `user_home_t`.

8.3 Contextos de Seguridad

Cada proceso y archivo en un sistema con SELinux activo tiene un **contexto de seguridad** que define su identidad dentro del modelo de control. Este contexto sigue la estructura:

```
usuario:rol:tipo:nivel
```

Por ejemplo:

```
kmacmill:staff_r:firefox_t:s0
kmacmill:object_r:user_home_t:s0
```

Estos contextos determinan las reglas aplicables según las políticas definidas por el administrador. El núcleo verifica constantemente que las operaciones entre contextos sean válidas según los permisos de acceso establecidos.

8.4 Mecanismos de Control en SELinux

SELinux implementa varios mecanismos de control complementarios que definen su arquitectura de seguridad:

- **Type Enforcement (TE):** (Se detalla más abajo)
- **RBAC (Role-Based Access Control):** Permite asociar roles a los usuarios, de modo que las operaciones disponibles dependen del rol activo, no del usuario individual.
- **MLS (Multi-Level Security):** Agrega una jerarquía de niveles de sensibilidad o clasificación (por ejemplo: público, confidencial, secreto), lo que impide el flujo de información de niveles altos a bajos (modelo Bell-LaPadula).

8.5 Type Enforcement (TE)

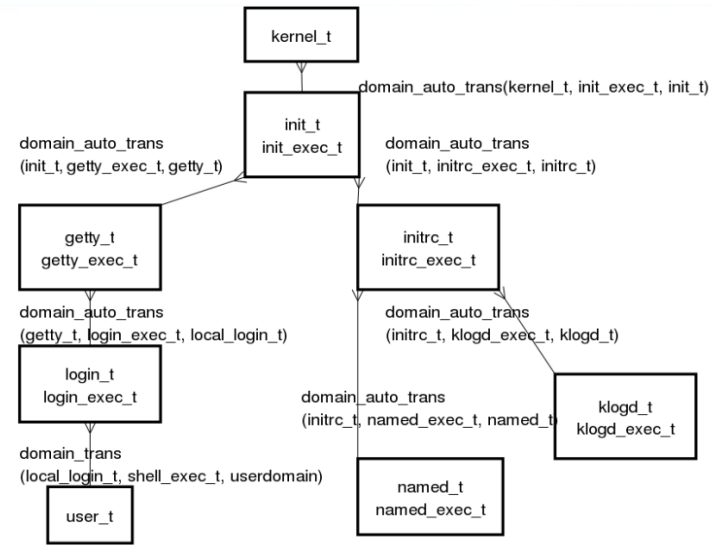
El **Type Enforcement (TE)** es el mecanismo primario de control en SELinux. Cada proceso y objeto del sistema tiene asociado un *tipo de seguridad*. Cuando el tipo se aplica a procesos, se lo denomina **dominio**. Estos tipos representan toda la información relevante de seguridad y definen qué interacciones son posibles entre procesos y recursos.

- Ejemplo de tipos:
  - Proceso del servidor web Apache: `httpd_t`
  - Archivo de contenido web: `/var/www/html/index.html` → `httpd_sys_content_t`
- El acceso entre tipos se define explícitamente mediante reglas, por ejemplo:

```
allow httpd_t httpd_sys_content_t : file read;
```

De esta manera, sólo las combinaciones de tipos autorizadas por las políticas podrán acceder a los recursos del sistema. Esto permite un confinamiento estricto de servicios y aplicaciones.

8.5.1 Decisiones y Transiciones



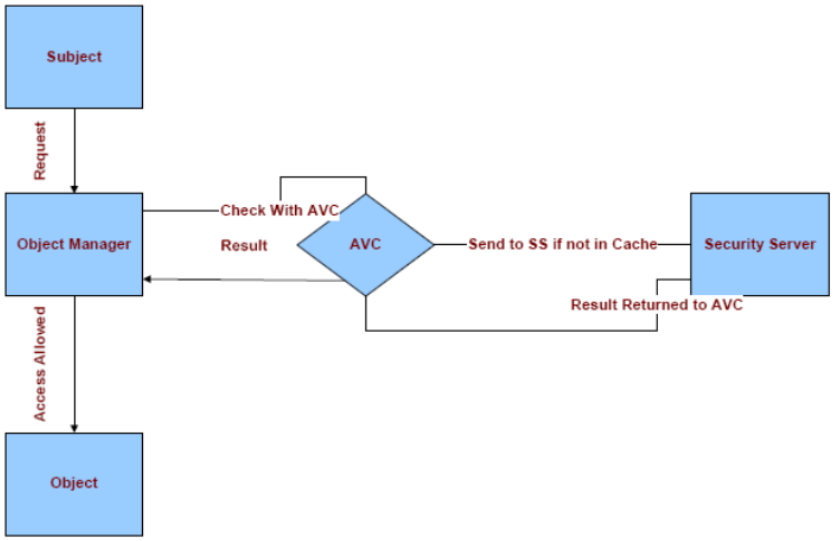
Transiciones de Proceso : Cuando un proceso crea otro, pueden ocurrir dos tipos de transición:

- **Herencia de dominio:** el proceso hijo mantiene el mismo dominio que su padre (por ejemplo, al ejecutar un comando desde un editor).
- **Cambio de dominio:** el proceso hijo adopta un nuevo dominio, definido por la política (por ejemplo, cuando `init` lanza un demonio `ftpd`).

Transiciones de Archivo: Cuando se crean archivos nuevos, éstos pueden:

- Heredar el tipo del directorio en el que se crean.
- Adoptar un tipo diferente según la política establecida (por ejemplo, los archivos temporales generados por un proceso pertenecen a un tipo definido como `tmp_t`).

8.5.2 Toma de Decisiones



Cada objeto del sistema posee un **Security Context (SC)** que tiene la forma general:

```
usuario:rol:tipo:nivel
```

Por ejemplo:

```
system_u:object_r:inetd_exec_t:s0
```

El campo clave es el **tipo**, que determina las reglas de acceso entre sujetos y objetos.
El **Security Server** (SS) asigna un contexto a cada proceso utilizando:

- Las reglas definidas en la política.
- El contexto del proceso padre.
- El identificador de usuario.

El servidor de seguridad evalúa cada intento de acceso consultando las políticas de TE. A diferencia del Unix estándar, donde los permisos se verifican solo al abrir un archivo, en SELinux cada operación se valida dinámicamente.

Para optimizar este proceso, SELinux utiliza el **Access Vector Cache (AVC)**, que almacena las decisiones de acceso previamente calculadas y permite responder rápidamente a solicitudes repetidas.

8.5.3 Implementación

SELinux mantiene la misma estructura funcional independientemente de la distribución Linux utilizada. Lo que varía entre distribuciones es:

- La forma de especificar y configurar las reglas.

- Las políticas incluidas por defecto.
- Las herramientas y comandos compatibles con SELinux.

El núcleo de SELinux se implementa como un módulo de seguridad (*Linux Security Module*, LSM), mientras que las distribuciones proveen distintos niveles de integración mediante utilitarios y configuraciones predeterminadas.

8.5.4 Componentes del Sistema

Aplicaciones

La mayoría de las aplicaciones y servicios no requieren modificaciones para ejecutarse en sistemas SELinux. Sin embargo, existen aplicaciones denominadas *SELinux-aware*, que son capaces de manipular o consultar contextos de seguridad. Ejemplos incluyen: `login`, `sshd`, `ls`, `cp`, `ps`, `setfilecon`, `logrotate`, `cron`, entre otros.

Políticas

Las políticas determinan el comportamiento general de SELinux. Existen dos enfoques principales:

Política Strict

- Todo acceso está denegado por defecto.
- Es necesario definir reglas explícitas para permitir privilegios.
- Diseñada bajo el principio de privilegio mínimo.
- No se manejan reglas de denegación explícitas ("deny"); todo lo no permitido está implícitamente bloqueado.
- Adecuada para entornos altamente controlados, pero difícil de mantener en sistemas de propósito general.

Política Targeted

- Alternativa práctica al modo *Strict*.
- Restringe principalmente a servicios críticos o potencialmente vulnerables (por ejemplo, `httpd`, `sshd`, `named`).
- Por defecto, los procesos corren en el dominio `unconfined_t`, con los mismos privilegios que un sistema sin SELinux.
- Los servicios con políticas definidas realizan una transición hacia un dominio confinado (por ejemplo, `httpd_t`) al iniciarse.

8.5.5 Dominios Predefinidos y Evolución

El número de dominios predefinidos aumentó significativamente entre versiones de RHEL:

- **RHEL 4:** alrededor de 15 dominios (por ejemplo: `httpd`, `squid`, `named`, `mysqld`, `syslogd`).
- **RHEL 5:** más de 200 dominios. Todo programa que se inicia al arrancar el sistema debe tener un dominio definido, lo que permite un control granular y una mejor segmentación de privilegios.

8.5.6 Archivos de Configuración

La configuración principal de SELinux se encuentra en el directorio `/etc/selinux`:

```
ls -l /etc/selinux
-rw-r--r-- 1 root root 515 Jan 18 11:46 config
drwxr-xr-x 7 root root 4096 Jan 23 14:06 strict
drwxr-xr-x 7 root root 4096 Jan 23 14:06 targeted
```

El archivo principal de configuración es `/etc/selinux/config`, donde se define el tipo de política y el modo operativo del sistema:

```
# This file controls the state of SELinux on the system.
SELINUX=enforcing
SELINUXTYPE=targeted
```

8.5.7 Modos de Operación

SELinux puede funcionar en tres modos principales:

- **Enforcing:** las políticas se aplican y se bloquean las operaciones no permitidas.
- **Permissive:** las violaciones de políticas se registran, pero no se bloquean.
- **Disabled:** SELinux no está activo ni carga políticas.

8.6 Comandos y Utilitarios de SELinux

SELinux extiende muchas de las utilidades tradicionales de Linux con soporte para contextos de seguridad, permitiendo administrar políticas, auditar accesos y configurar el sistema sin comprometer la seguridad.

8.6.1 Comandos Modificados

En SELinux, el modificador `-Z` es fundamental para consultar y establecer el contexto de seguridad (*security context*) de archivos y procesos. Por eso se dice que: "**Z es tu amigo!**".

Algunas de las utilidades principales adaptadas con soporte de contextos incluyen:

- **Core Utilities:**
  - `ls -Z` : muestra el contexto de seguridad de archivos y directorios.
  - `cp -Z`, `mv -Z`, `install -Z` : preserva o asigna contextos al copiar o mover archivos.
  - `find / -context=` : busca archivos según su contexto de seguridad.
  - `id -Z` : muestra el contexto de seguridad del usuario actual.
  - `ps auxZ` : muestra los procesos en ejecución con su contexto SELinux.
  - `netstat -Z` : muestra conexiones de red con los contextos de seguridad asociados.
- **Login, PAM y gestión de usuarios:**
  - `ssh`, `su`, `login`, `xdm`, `sudo` : todos interactúan con los contextos de usuario y roles.
  - `passwd`, `useradd`, `groupadd` : modifican contextos de seguridad asociados a cuentas.
- **RPM:** los paquetes también gestionan etiquetas de contexto SELinux al instalar o actualizar software.

8.6.2 Utilitarios SELinux

SELinux incluye un conjunto de utilitarios específicos para administración y auditoría:

- **Policycoreutils:**
  - `newrole` : cambia el rol del usuario actual según la política.
  - `run_init` : inicia procesos en contextos específicos.
  - `audit2allow`, `audit2why` : generan reglas y explicaciones a partir de mensajes de auditoría.
  - `secon` : muestra información sobre el servidor de seguridad.
  - `sestatus` : muestra el estado actual de SELinux (enforcing, permissive, disabled) y la política cargada.
- **Libselinux:**
  - `getenforce` / `setenforce` : consulta y modifica el modo de operación de SELinux.
  - `selinuxenabled` : verifica si SELinux está activo en el sistema.

8.6.3 Manejo de Booleans y Contextos de Archivos

Booleans de SELinux. Los booleans permiten modificar el comportamiento de la política de forma condicional, similar a instrucciones *if/then/else*, sin necesidad de editar directamente los archivos de política:

- `getsebool -a` : muestra todos los booleanos y su estado actual.
- `setsebool -P httpd_enable_homedirs 1` : activa o desactiva booleanos de manera persistente.

Contextos de archivos. SELinux también provee utilitarios para gestionar contextos de seguridad de archivos:

- `setfiles`, `restorecon`, `fixfiles`, `genhomedircon`, `chcon` : permiten asignar, restaurar o corregir contextos según la política vigente.

8.6.4 Auditoría y generación de políticas

SELinux proporciona herramientas para generar reglas basadas en eventos registrados en los logs de auditoría:

- `audit2allow -i /var/log/audit/audit.log` : genera reglas de acceso que permiten las operaciones bloqueadas.
- Ejemplo de regla generada:

```
allow system_crond_t null_device_t:chr_file { read write };
```

- `audit2why` : traduce mensajes de auditoría en explicaciones legibles sobre por qué se denegaron accesos.

8.6.5 Httpd y Booleans específicos

Para servicios como Apache (`httpd`), los booleans permiten configurar la política sin editarla manualmente:

- `getsebool -a` : verifica los booleans activos para httpd y otros servicios.
- `setsebool -P httpd_enable_homedirs 1` : permite a httpd acceder a directorios de usuario, modificando la política dinámicamente.

Estos booleans se documentan en el manual específico del servicio, por ejemplo: `man httpd_selinux`.

8.7 SELinux GUI y Cambio de Contexto de Archivos

SELinux no solo se administra mediante la línea de comandos; también existen herramientas gráficas que facilitan la configuración y supervisión de políticas y contextos de seguridad.

8.7.1 SELinux GUI – system-config-selinux

system-config-selinux es una interfaz gráfica (GUI) para la administración de SELinux en sistemas Linux. Sus características principales incluyen:

- Permite activar o desactivar SELinux y seleccionar la política a utilizar (`targeted` o `strict`).
- Facilita la visualización de contextos de seguridad de archivos, procesos y servicios.
- Proporciona herramientas para modificar roles, tipos y niveles de seguridad sin necesidad de interactuar directamente con los archivos de configuración del sistema.
- Ofrece paneles para configurar booleanos de SELinux de forma interactiva.

La GUI es especialmente útil para administradores que prefieren una gestión visual del sistema o para entornos donde los cambios frecuentes en políticas requieren una supervisión rápida.

8.7.2 Cambio de Contexto de Archivos

En SELinux, los archivos poseen un contexto de seguridad que determina quién puede acceder a ellos y cómo. Cambiar estos contextos se realiza principalmente mediante la utilidad `chcon`.

Uso básico de `chcon`

`chcon` permite modificar el tipo, rol, usuario o nivel de un archivo o directorio, de manera similar a cómo `chmod` cambia permisos:

- `chcon -t httpd_sys_script_rw_t /var/www/myapp/data` Cambia el tipo del directorio de datos de una aplicación web para que el servidor Apache pueda leer y escribir.
- `chcon -t httpd_sys_script_t /var/www/cgi-bin/myapp` Aplica un tipo específico a scripts ejecutables en `cgi-bin`.

Opciones importantes

- `-R`: aplica cambios recursivamente a todos los subdirectorios y archivos.
- `-t type`: permite especificar el tipo de seguridad que se asignará.

Personalización de tipos

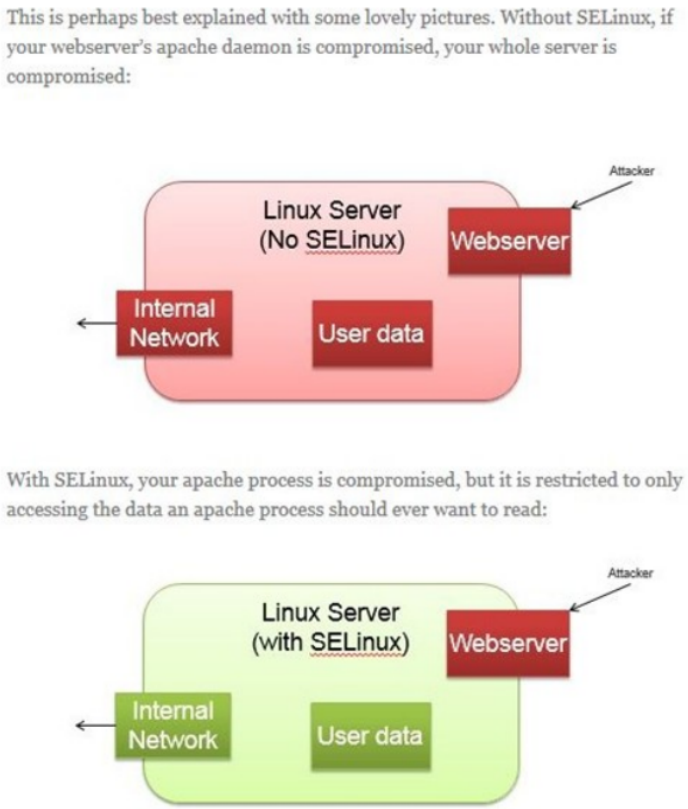
SELinux permite definir tipos personalizables que se pueden consultar y editar en el archivo:

`/etc/selinux/targeted/contexts/customizable_types`

Esto facilita crear contextos específicos para aplicaciones o servicios que no estén cubiertos por los tipos estándar, garantizando compatibilidad con la política activa sin comprometer la seguridad.

Al modificar contextos de archivos o directorios, se debe tener especial cuidado, ya que una asignación incorrecta podría permitir que procesos accedan a recursos que deberían estar aislados. Es importante seguir las recomendaciones de la política SELinux vigente y probar los cambios en entornos controlados antes de aplicarlos en producción.

8.7.3 Shellshock



Shellshock es una vulnerabilidad crítica descubierta en 2014 en **Bash**, el shell más común en Linux y sistemas Unix. Su nombre oficial es `CVE-2014-6271` y tiene múltiples variantes que fueron corregidas posteriormente. Esta vulnerabilidad permite a un atacante ejecutar **comandos arbitrarios** manipulando variables de entorno.

Funcionamiento

1. Bash permite definir **funciones en variables de entorno**. Ejemplo:

   ```
   export FOO='() { echo hello; }'
   ```

2. La vulnerabilidad ocurre cuando Bash **evalúa estas funciones** al iniciar un shell y ejecuta código adicional no autorizado:

   ```
   export FOO='() { :;}; /usr/bin/id'
   ```

En sistemas vulnerables, el comando `/usr/bin/id` se ejecuta automáticamente.

3. Si un servicio expone variables de entorno desde el exterior (por ejemplo, scripts CGI en un servidor web), un atacante puede **inyectar código malicioso** y ejecutar comandos con los permisos del proceso afectado.

8.7.4 Por qué es peligroso?

- Afecta servidores web, scripts CGI, cron jobs y servicios que usan Bash para procesar variables de entorno.
- Permite que un atacante tome control del sistema, ejecute malware o robe información.
- Incluso sistemas protegidos por SELinux pueden verse comprometidos si los contextos de seguridad permiten que Bash ejecute scripts con permisos elevados.

SELinux puede **confinar procesos vulnerables** (por ejemplo, servidores web) a dominios específicos, limitando los archivos y recursos a los que pueden acceder. Esto **reduce el riesgo de explotación**, aunque no elimina la vulnerabilidad en Bash. Por ello, SELinux protege la integridad y el aislamiento, pero no reemplaza la necesidad de parches.