FACULTAD DE CIENCIAS EXACTAS, FÍSICAS y NATURALES

SISTEMAS DE COMPUTACIÓN

TP4: Módulos de kernel y llamadas a sistema

Nombre	DNI	Carrera
Facundo Nahuel Galvagno	40815088	IE
Tomás Ignacio Corbalán	38503847	ICOMP
Hernan Imanol Rodriguez	41808427	ICOMP

Desafío #1

Checkinstall

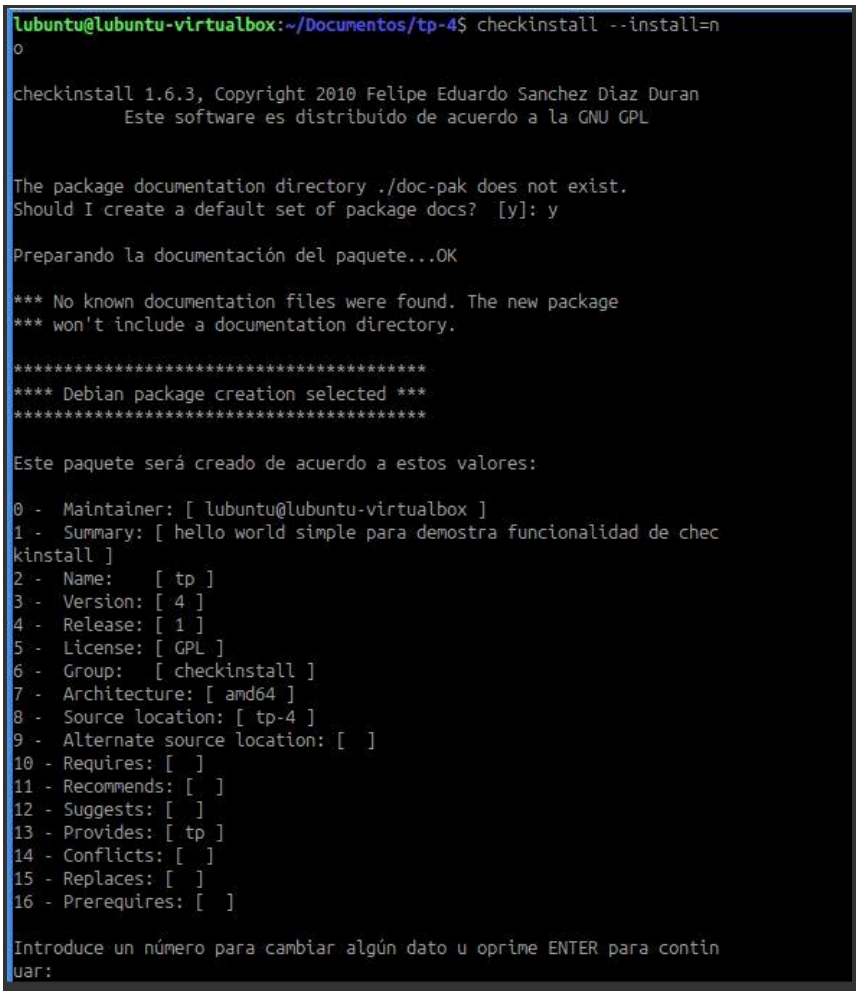
Es una utilidad para gestionar paquetes y crear autoinstalables clásicos de Linux .rpm / .deb

Se instala con (Fedora) descargando el .rpm de

https://fedora.pkgs.org/41/rpm-sphere-x86_64/checkinstall-1.6.2-1.x86_64.rpm.html

luego, se debe desarrollar el software que sera empaquetado

```
make
sudo checkinstall --install=no
```



Cómo Evitar la Carga de Módulos de Kernel No Firmados

La capacidad de firmar digitalmente los módulos de kernel es una característica de seguridad fundamental en Linux para asegurar que solo el código confiable y autorizado se cargue en el kernel. Aquí se detallan los pasos y configuraciones para evitar la carga de módulos no firmados:

1. **Habilitar la Verificación de Firma del Módulo en el Kernel (Kernel Module Signature Verification):** Esta es la medida más importante. El kernel de Linux debe estar compilado con las opciones de configuración adecuadas para habilitar la verificación de firma. Las opciones relevantes son:

- `CONFIG_MODULE_SIG=y`: Habilita la infraestructura de firma de módulos.
- `CONFIG_MODULE_SIG_FORCE=y`: **Esta opción es crucial.** Si está habilitada, el kernel rechazará la carga de cualquier módulo que no esté firmado o cuya firma sea inválida. Sin esta opción, la verificación de firma solo emitirá advertencias, pero permitirá la carga del módulo.
- `CONFIG_MODULE_SIG_ALL=y`: Firma todos los módulos en el momento de la compilación del kernel.
- `CONFIG_MODULE_SIG_HASH="sha256"` (o sha512): Define el algoritmo de hash utilizado para las firmas.
- `CONFIG_SYSTEM_TRUSTED_KEYS`: Define el conjunto de claves públicas confiables que el kernel usará para verificar las firmas. Estas claves se suelen incorporar directamente en el binario del kernel.

Verificación de la Configuración del Kernel: Puedes verificar si tu kernel actual fue compilado con estas opciones ejecutando:

```
grep CONFIG_MODULE_SIG /boot/config-$(uname -r)
```

- Si `CONFIG_MODULE_SIG_FORCE=y` no está presente o está configurado como `n`, tu kernel permitirá módulos no firmados. En ese caso, necesitarías recompilar el kernel con estas opciones habilitadas, o usar un kernel proporcionado por tu distribución que ya las tenga activadas.

- **Generación y Gestión de Claves de Firma:** Para firmar tus propios módulos (si desarrollas o compilas tus propios módulos legítimos), necesitas un par de claves (privada y pública).

- La clave privada se usa para firmar el módulo.
- La clave pública se incorpora en el kernel o en el `_keyring_` del sistema para verificar la firma.

El proceso general implica:

- Crear un par de claves X.509.
- Compilar el kernel, asegurándote de que las claves confiables se añadan al `_keyring_` del kernel.
- Firmar los módulos después de su compilación utilizando la clave privada. Las herramientas de compilación del kernel (`kbuild`) generalmente se encargan de esto si `CONFIG_MODULE_SIG_ALL=y` está habilitado. Si firmas manualmente, puedes usar `scripts/sign-file`.

- **Restricciones de Carga de Módulos (Secure Boot y Módulos de Kernel)**:

-`Secure Boot:` Si el sistema soporta y tiene habilitado UEFI Secure Boot, se asegura que solo el cargador de arranque (bootloader) y el kernel firmados digitalmente por una autoridad de confianza puedan iniciarse. Esto es un paso previo a la carga de módulos, pero fundamental. Si un atacante no puede arrancar un kernel modificado, se reduce significativamente el riesgo.

- `modprobe` y Permisos: Aunque `modprobe` y `insmod` requieren privilegios de root para cargar módulos, un atacante con acceso de root podría intentar cargar un módulo malicioso. La verificación de firma actúa como una segunda línea de defensa, incluso si el atacante obtiene privilegios de root.

Definición y Clasificación de Rootkits

Un rootkit es un tipo de software malicioso caracterizado por su sigilo, diseñado para ocultar su propia presencia y mantener un acceso privilegiado y persistente en un sistema que ha sido comprometido. Es importante destacar que un rootkit no es un vector de ataque inicial; en cambio, es una herramienta de post-explotación. Su propósito es establecer persistencia, escalar privilegios y facilitar la vigilancia una vez que un atacante ya ha obtenido acceso al sistema.

Los rootkits se clasifican según el nivel de control que ejercen sobre el sistema:

- Rootkits en Modo Usuario (User-Mode Rootkits): Operan en el espacio de usuario, modificando el comportamiento de las aplicaciones (por ejemplo, la enumeración de archivos o el listado de procesos) sin alterar directamente el kernel del sistema operativo. Son relativamente más sencillos de desplegar, pero también más susceptibles de ser detectados por herramientas de Detección y Respuesta de Puntos Finales (EDR). Para lograr su objetivo, emplean técnicas como la inyección de DLL y el enganche de API, incluyendo el enganche de la Tabla de Direcciones de Importación (IAT hooking) y el enganche en línea (inline hooking).

- Rootkits en Modo Kernel (Kernel-Mode Rootkits): Estos son los más peligrosos y elusivos. Se incrustan a nivel del kernel del sistema operativo, enganchando o parcheando llamadas al sistema. Esto les otorga acceso irrestricto al hardware, a los controladores y a los controles de seguridad del sistema. Al modificar directamente los módulos del kernel, pueden ocultar archivos, sockets y procesos tanto de los usuarios como del software de defensa. Son considerados los más difíciles de detectar.

- Rootkits de Firmware: Infectan componentes de bajo nivel como la BIOS, UEFI o el firmware de dispositivos periféricos (tarjetas de red, unidades de disco). Debido a que residen fuera del sistema operativo, son invisibles para los agentes de seguridad de los puntos finales y persisten incluso después de reinicios o reemplazos completos del disco. La recuperación de estos rootkits a menudo requiere el reflasheo del hardware.

- Rootkits de Hipervisor: Explotan las capacidades de virtualización del hardware para ejecutar el sistema operativo objetivo dentro de una capa virtualizada que es controlada por el atacante. Aunque el sistema operativo anfitrión parece funcionar normalmente, toda su actividad es monitoreada y puede ser modificada por el rootkit

- Rootkits de Librería: Sustituyen o secuestran bibliotecas de sistema compartidas (por ejemplo, libc en Linux o DLLs en Windows) para alterar el comportamiento de los programas, filtrar salidas, enmascarar archivos o redirigir llamadas al sistema. Son generalmente más fáciles de detectar mediante comprobaciones de integridad.

Tecnologías de protección contra rootkits:

entre las distintas soluciones exploramos 2 sistemas de protección, AppArmor y SELinux:

1. **AppArmor**

¿Qué es?

AppArmor es un sistema de **Security Modules (LSM)** que restringe aplicaciones mediante **perfiles de seguridad**, definiendo qué archivos, capacidades y recursos puede acceder cada proceso.

Características Clave:

- **Basado en rutas**: Los perfiles usan rutas de archivos explícitas (`/usr/bin/firefox`).
- **Modos**:
  - **Enforce**: Bloquea acciones no permitidas.
  - **Complain**: Solo registra violaciones (sin bloquear).
- **Perfiles predefinidos**: Vienen con aplicaciones comunes (Apache, Firefox).

2.**SELinux**

¿Qué es?

SELinux (Security-Enhanced Linux) es un sistema de mandatory access control (MAC) que usa contextos de seguridad (etiquetas) para definir políticas granulares.

Características Clave:

- **Basado en etiquetas: Cada archivo/proceso tiene un contexto (`user_u:role_r:type_t`).**
- **Modos:**
  - **Enforcing: Bloquea acciones no permitidas.**
  - **Permissive: Solo registra violaciones.**
  - **Disabled: Desactivado.**
- **Políticas estrictas: Define reglas para usuarios, roles y tipos.**

Comparativa AppArmor vs SELinux

Característica	AppArmor (Debian/Ubuntu)	SELinux (RHEL/Fedora)
Enfoque	Basado en rutas	Basado en etiquetas (contextos)
Facilidad	Más fácil de configurar	Curva de aprendizaje pronunciada
Políticas	Perfiles por aplicación	Reglas granulares (usuarios/roles)
Comandos clave	`apparmor_status, aa-enforce`	`getenforce, semanage, ls -Z`
Uso típico	Escritorio, servidores simples	Entornos empresariales/seguros

Desafío #2

Debe tener respuestas precisas a las siguientes preguntas y sentencias:

¿Qué funciones tiene disponible un programa y un módulo?

Un programa se ejecuta en el espacio de usuario y tiene acceso a:

- Funciones de biblioteca estándar (como printf, malloc, fopen, etc.).
- Llamadas al sistema (syscalls) como read, write, open, fork, exec, exit.
- APIs del sistema operativo, proporcionadas por bibliotecas como libc o POSIX.
- Funciones propias definidas en el código fuente del programa.
- Importante: No puede acceder directamente al hardware o memoria del sistema, todo debe pasar por el kernel (mediante llamadas al sistema o drivers).

Un módulo del kernel (como un módulo de Linux con extensión .ko) se ejecuta en modo kernel y tiene acceso a:

- Funciones internas del kernel, como copy_to_user, printk, kmalloc, request_irq, etc.
- Funciones exportadas por otros módulos o por el núcleo (EXPORT_SYMBOL).
- Manejo directo de hardware y recursos (acceso a puertos, registros, interrupciones).
- Gestión de procesos, memoria, dispositivos, redes, etc.
- Importante: El código del kernel es crítico; errores pueden colapsar el sistema.

Espacio de usuario o espacio del kernel.

Característica	Espacio de Usuario	Espacio del Kernel
¿Quién lo usa?	Programas de usuario (como editores, apps)	Sistema operativo y módulos del kernel
Privilegios	Limitados (protegido)	Privilegiado (acceso completo)
Seguridad	Aislado, no puede dañar el sistema	Errores pueden afectar todo el sistema
Acceso al hardware	No directo, solo mediante llamadas al SO	Directo
Protección de memoria	Sí	No entre partes del kernel

Espacio de datos.

El espacio de datos de un programa se refiere a las zonas de memoria que contienen variables y datos en tiempo de ejecución. Se divide en:

- Data segment: variables globales y estáticas inicializadas.
- BSS segment: variables globales/estáticas no inicializadas.
- Heap: memoria dinámica (usada por malloc, new, etc.).
- Stack: almacena variables locales y llamadas a funciones.

En el kernel también existe un espacio de datos propio, utilizado para sus estructuras internas.

Drivers. Investigar contenido de /dev.

El directorio /dev contiene archivos especiales de dispositivo, que representan interfaces para acceder a drivers desde el espacio de usuario.

Tipos de dispositivos en /dev:

- Dispositivos de carácter (character devices): accedidos byte a byte (/dev/tty, /dev/random, /dev/null).
- Dispositivos de bloque (block devices): accedidos por bloques (/dev/sda, /dev/mmcblk0).
- Dispositivos pseudo: interfaces especiales como /dev/zero, /dev/full, /dev/loop*.

Estos archivos permiten a los programas acceder a hardware como si fuera un archivo normal (con read, write, ioctl, etc.).

```
facundog@FacundoNotebook:/dev$ ls
autofs           hwrng     loop4         rtc          tty14  tty39  tty63     ttyS29       vcs4
block            i2c-0     loop5         rtc0         tty15  tty4   tty7      ttyS3        vcs5
bsg              i2c-1     loop6         sda          tty16  tty40  tty8      ttyS30       vcs6
btrfs-control    i2c-2     loop7         sda1         tty17  tty41  tty9      ttyS31       vcsa
bus              i2c-3     loop8         sda2         tty18  tty42  ttyprintk ttyS4        vcsa1
char             i2c-4     loop9         sdb          tty19  tty43  ttyS0     ttyS5        vcsa2
console          i2c-5     loop-control  sdb1         tty2   tty44  ttyS1     ttyS6        vcsa3
core             i2c-6     mapper        sg0          tty20  tty45  ttyS10    ttyS7        vcsa4
cpu              initctl   mcelog        sg1          tty21  tty46  ttyS11    ttyS8        vcsa5
cpu_dma_latency  input     media0        sgx_enclave  tty22  tty47  ttyS12    ttyS9        vcsa6
cuse             kmsg      mei0          sgx_provision tty23 tty48  ttyS13    tuxedo_io    vcsu
disk             kvm       mem           sgx_vepc     tty24  tty49  ttyS14    udmabuf      vcsu1
dma_heap         log       mqueue        shm          tty25  tty5   ttyS15    uhid         vcsu2
dri              loop0     mtd           snapshot     tty26  tty50  ttyS16    uinput       vcsu3
drm_dp_aux0      loop1     mtd0          snd          tty27  tty51  ttyS17    urandom      vcsu4
ecryptfs         loop10    mtd0ro        stderr       tty28  tty52  ttyS18    usb          vcsu5
fb0              loop11    net           stdin        tty29  tty53  ttyS19    userfaultfd  vcsu6
fd               loop12    null          stdout       tty3   tty54  ttyS2     userio       vfio
full             loop13    nvram         tpm0         tty30  tty55  ttyS20    v4l          vga_arbiter
fuse             loop14    port          tpmrm0       tty31  tty56  ttyS21    vboxdrv      vhci
gpiochip0        loop15    ppp           tty          tty32  tty57  ttyS22    vboxdrvu     vhost-net
hidraw0          loop16    psaux         tty0         tty33  tty58  ttyS23    vboxnetctl   vhost-vsock
hidraw1          loop17    ptmx          tty1         tty34  tty59  ttyS24    vboxusb      video0
hidraw2          loop18    ptp0          tty10        tty35  tty6   ttyS25    vcs          video1
hidraw3          loop19    pts           tty11        tty36  tty60  ttyS26    vcs1         zero
hpet             loop2     random        tty12        tty37  tty61  ttyS27    vcs2         zfs
hugepages        loop3     rfkill        tty13        tty38  tty62  ttyS28    vcs3
facundog@FacundoNotebook:/dev$
```

Desafío #3

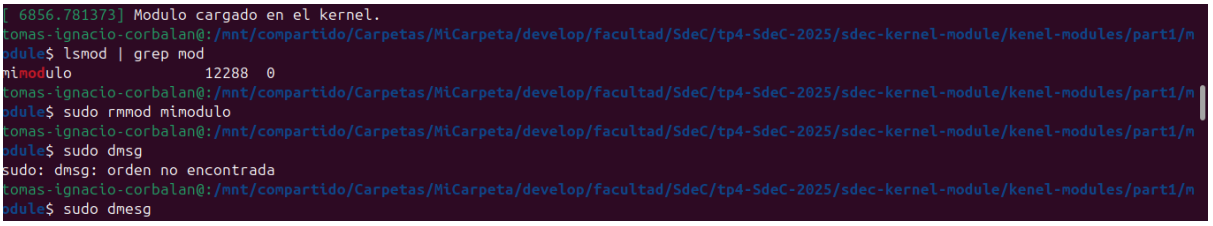
Realizamos la instalación del módulo de kernel mimodulo.ko luego de compilar el Makefile del repositorio de gitlab disponible para el tp4 previamente clonado:

```
sudo insmod mimodulo.ko
```

```
Terminal
odule$ ls
Makefile  mimodulo.c  mimodulo.ko  mimodulo.mod  mimodulo.mod.c  mimodulo.mod.o  mimodulo.o  modules.order  Module.symvers
tomas-ignacio-corbalan@:/mnt/compartido/Carpetas/MiCarpeta/develop/facultad/SdeC/tp4-SdeC-2025/sdec-kernel-module/kenel-modules/part1/m
odule$ sudo insmod mimodulo.ko
[sudo] contraseña para tomas-ignacio-corbalan:
tomas-ignacio-corbalan@:/mnt/compartido/Carpetas/MiCarpeta/develop/facultad/SdeC/tp4-SdeC-2025/sdec-kernel-module/kenel-modules/part1/m
odule$ sudo dmesg
[    0.000000] Linux version 6.11.0-25-generic (buildd@lcy02-amd64-027) (x86_64-linux-gnu-gcc-13 (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0,
 GNU ld (GNU Binutils for Ubuntu) 2.42) #25~24.04.1-Ubuntu SMP PREEMPT_DYNAMIC Tue Apr 15 17:20:50 UTC 2 (Ubuntu 6.11.0-25.25~24.04.1-g
eneric 6.11.11)
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-6.11.0-25-generic root=UUID=67a6b128-b526-421c-bc77-65a8cca0f766 ro quiet splash
vt.handoff=7
[    0.000000] KERNEL supported cpus:
[    0.000000]   Intel GenuineIntel
[    0.000000]   AMD AuthenticAMD
[    0.000000]   Hygon HygonGenuine
[    0.000000]   Centaur CentaurHauls
[    0.000000]   zhaoxin   Shanghai
[    0.000000] BIOS-provided physical RAM map:
```

Se puede observar el mensaje impreso en el kernel por nuestro programa con la evidencia de su correcta instalación.
Luego procedemos a remover el módulo del kernel mediante el siguiente comando:

```
sudo rmmod mimodulo
```



Se puede observar el mensaje impreso por nuestro programa con la evidencia de que el módulo ha sido retirado del kernel:

```
[ 6856.781373] Modulo cargado en el kernel.
[ 7109.647665] Modulo descargado del kernel.
tomas-ignacio-corbalan@:/mnt/compartido/Carpetas/MiCarpeta/develop
/facultad/SdeC/tp4-SdeC-2025/sdec-kernel-module/kenel-modules/part
1/module$ lsmod | grep mod
tomas-ignacio-corbalan@:/mnt/compartido/Carpetas/MiCarpeta/develop
/facultad/SdeC/tp4-SdeC-2025/sdec-kernel-module/kenel-modules/part
1/module$ cat /proc/modules | grep mod
```

1. **¿Qué diferencias se pueden observar entre los dos modinfo?**

Ejecutamos `modinfo mimodulo.ko` :

```
tomas-ignacio-corbalan@:/mnt/compartido/Carpetas/MiCarpeta/develop
/facultad/SdeC/tp4-SdeC-2025/sdec-kernel-module/kenel-modules/part
1/module$ modinfo mimodulo.ko
filename:       /mnt/compartido/Carpetas/MiCarpeta/develop/faculta
d/SdeC/tp4-SdeC-2025/sdec-kernel-module/kenel-modules/part1/module
/mimodulo.ko
author:         Catedra de SdeC
description:    Primer modulo ejemplo
license:        GPL
srcversion:     C6390D617B2101FB1B600A9
depends:
retpoline:      Y
name:           mimodulo
vermagic:       6.11.0-25-generic SMP preempt mod_unload modversio
ns
```

Este módulo **no realiza una tarea funcional** del sistema operativo. Es un módulo de propósito **didáctico**, que imprime un mensaje al cargarse y otro al descargarse (como se ve en los logs).

Ejecutamos `modinfo /lib/modules/$(uname -r)/kernel/crypto/des_generic.ko` :

```
tomas-ignacio-corbalan@:/mnt/compartido/Carpetas/MiCarpeta/develop
/facultad/SdeC/tp4-SdeC-2025/sdec-kernel-module/kenel-modules/part
1/module$ modinfo /lib/modules/$(uname -r)/kernel/crypto/des_gener
ic.ko
modinfo: ERROR: Module /lib/modules/6.11.0-25-generic/kernel/crypt
o/des_generic.ko not found.
tomas-ignacio-corbalan@:/mnt/compartido/Carpetas/MiCarpeta/develop
/facultad/SdeC/tp4-SdeC-2025/sdec-kernel-module/kenel-modules/part
1/module$ find /lib/modules/$(uname -r) -name 'des_generic.ko*'
/lib/modules/6.11.0-25-generic/kernel/crypto/des_generic.ko.zst
```

Vemos que nos devuelve un ERROR diciendo que el Module **des_generic.ko** no fue encontrado y luego cuando ejecutamos:

```
find /lib/modules/$(uname -r) -name 'des_generic.ko*'
```

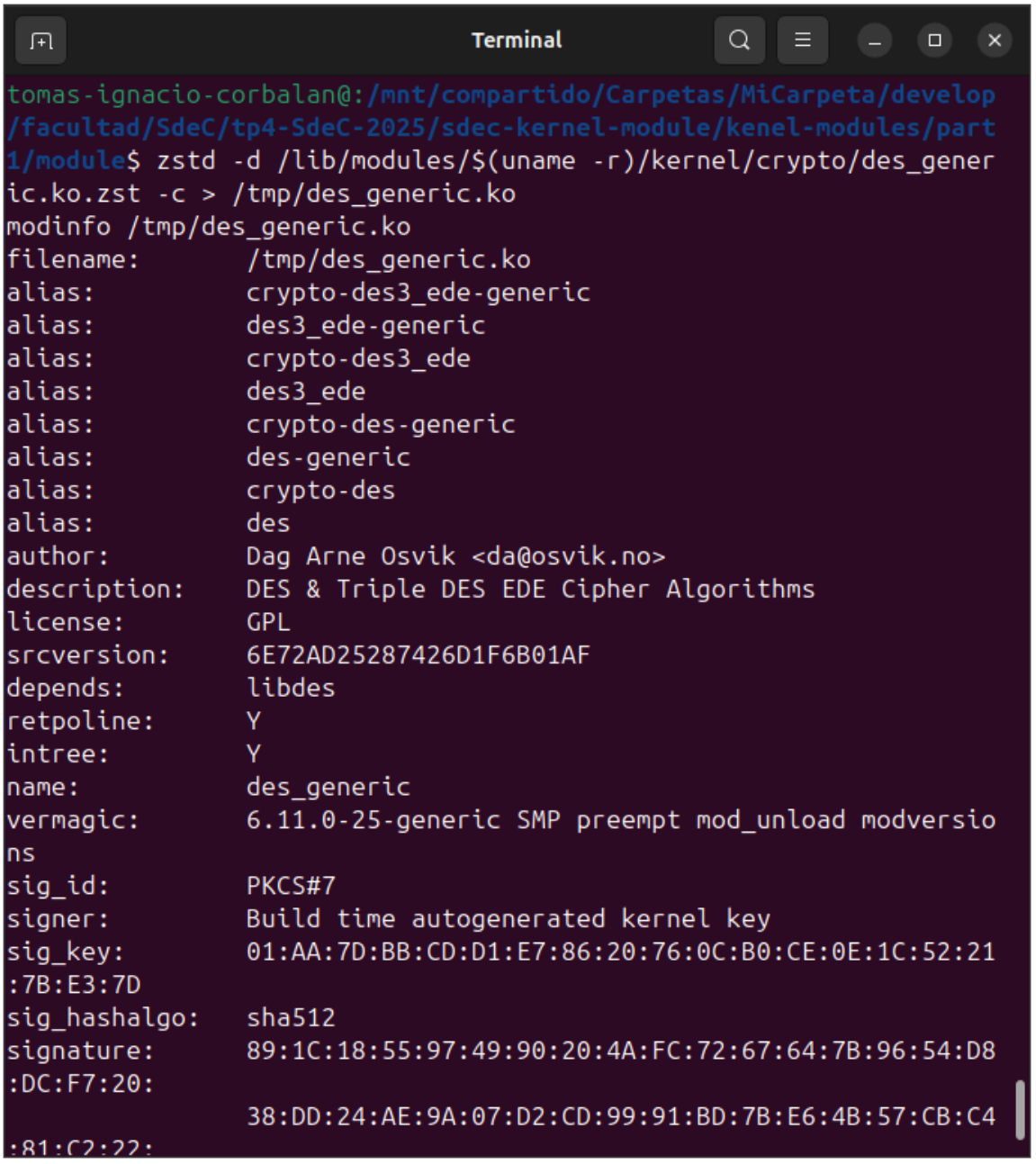
Vemos que encuentra a **des_generic.ko.zst:**

```
/lib/modules/6.11.0-25-generic/kernel/crypto/des_generic.ko.zst
```

Esto significa que el módulo si está instalado pero se encuentra comprimido con **Zstandard** (.zst) para ahorrar espacio. Para ver información del módulo podemos descomprimirlo temporalmente con **zstd** y pasarselo a **modinfo** para que esta vez sí pueda leer su información:

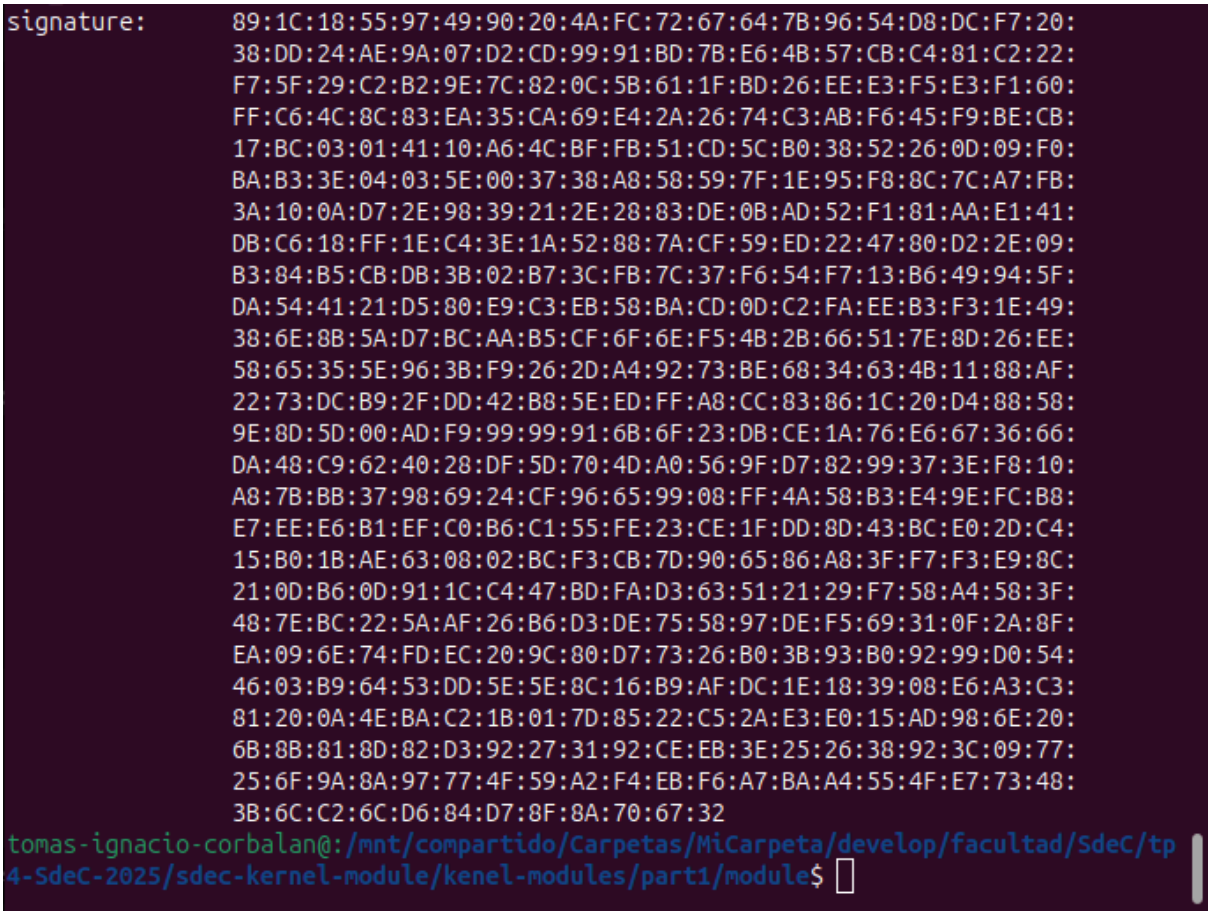
```
zstd -d /lib/modules/$(uname -r)/kernel/crypto/des_generic.ko.zst -c >
/tmp/des_generic.ko modinfo /tmp/des_generic.ko
```

Obteniendo la siguiente salida en la terminal:



Este módulo forma parte del **subsistema de criptografía del kernel**, e implementa los algoritmos de cifrado **DES** y **Triple DES**. Es un módulo del sistema base y puede ser utilizado por sistemas de archivos cifrados, VPNs, etc.

Podemos observar la firma del módulo:



Entonces **¿Qué diferencias se pueden observar entre los dos modinfo para ambos módulos?**. La diferencia principal radica en todos los campos relacionados con la firma de un módulo, en donde se puede observar que el módulo **des_generic.ko** está firmado y **mimodulo.ko** no lo está. La firma de módulos del kernel es una medida de seguridad muy importante ya que garantiza que el módulo fue generado por una fuente confiable y verifica la integridad del mismo asegurando que no ha sido modificado (bien o mal intencionadamente) desde que fue firmado. Se complementa con **Secure Boot**, en donde el kernel sólo carga módulos firmados.

Campos relacionados con la firma:

- sig_id: PKCS#7:
  - Formato de firma.
  - PKCS#7 es un estándar criptográfico de firma y cifrado de mensajes (usado también en correos seguros y certificados SSL).
  - Es el contenedor que agrupa firma, certificado y algoritmo hash.

- signer: Build time autogenerated kernel key:
  - Este campo indica el firmante del módulo.
  - En este caso: una clave generada automáticamente durante el build del kernel (por ejemplo, cuando Ubuntu construyó el kernel 6.11.0-25-generic).
  - En distros como Ubuntu, Fedora o Debian, se genera una clave privada para firmar los módulos y una pública que se instala en el kernel para verificar.

- sig_key: 01:AA:7D:BB:CD:...:1C:52:21:
  - El identificador de la clave pública usada para la firma.
  - Es una huella digital (fingerprint) de la clave, no la clave en sí.
  - Sirve para identificar la clave usada (útil si hay varias claves cargadas en el sistema para verificar firmas).

- sig_hashalgo: sha512
  - El algoritmo hash usado para firmar el contenido del módulo.
  - En este caso, SHA-512, una variante segura de la familia SHA-2.
  - Se usa para calcular un resumen (hash) del archivo del módulo antes de firmarlo.

- signature:
  - Esta es la firma digital completa, en formato hexadecimal.
  - Se genera al aplicar la clave privada al hash calculado del módulo.
  - El kernel, al cargar el módulo, verifica esta firma usando la clave pública que tiene cargada en su sistema (por ejemplo, en /proc/keys o compilada dentro del kernel).

2. ¿Qué drivers/módulos están cargados en sus propias pc? Comparar las salidas con las computadoras de cada integrante del grupo. Expliquen las diferencias. Carguen un txt con la salida de cada integrante en el repo y pongan un diff en el informe.

Módulos exclusivos de cada PC

PC1 (Facundo):

- Carga módulos de VirtualBox (`vboxdrv`, `vboxnetflt`, `vboxnetadp`) y NTFS3.
- Usa módulos relacionados a NVIDIA (`nvidia`, `nvidia_uvm`, `nvidia_drm`, `nvidia_modeset`) y audio con SOF (Sound Open Firmware).
- Tiene módulos para compresión de audio, varios relacionados al sistema de sonido HDA/HDMI, y también módulos para redes y virtualización (`kvm_intel`, `kvm`, etc.).
- Módulo personalizado llamado `mimodulo`.

PC2 (Tomás):

- Tiene soporte para Bluetooth extendido (`ath3k`, `btusb`, `btintel`, `btmtk`, etc.).
- Carga controladores Wi-Fi antiguos, como `ath9k`, `ath9k_common`, `ath9k_hw`.
- Tiene una configuración más liviana: no tiene VirtualBox, ni NVIDIA, ni módulos de audio avanzados.

PC3 (Imanol):

- Tiene soporte de VirtualBox al igual que PC1.

- Usa sonido USB (`snd_usb_audio`, `snd_usbmidi_lib`) en lugar de HDA.
- Tiene módulos de cámara tipo GSPCA (`gspca_ov534`, `gspca_main`) y soporte multimedia alternativo.
- Soporte de red y firewall avanzado con `nft_*`, `nf_conntrack_*`, `ip_set`.

Módulos comunes (aunque no siempre con la misma configuración):

- `uvcvideo`, `videobuf2_vmalloc`, `videodev`: manejo de cámaras/webcams.
- `qrtr`, `binfmt_misc`, `coretemp`, `intel_rapl_msr`, `snd_hda_codec_realtek`: comunes entre al menos dos PCs.
- `rfcomm`, `cmac`, `algif_hash`, `af_alg`: módulos de criptografía y Bluetooth compartidos entre PC1 y PC2.

3. Como revisar los módulos que no han sido cargados:

```
find /lib/modules/$(uname -r) -type f -name '*.ko' | sed s|.*/||;s|\.ko$||
```

este comando busca entre los módulos disponibles de acuerdo a la version de kernel actual

4. Correr hw-probe en una pc real con hw real y agregar la url de la información de hw-probe en el reporte.

Tomás Ignacio Corbalán: https://linux-hardware.org/?probe=e6d21eab37
Facundo Nahuel Galvagno: https://linux-hardware.org/?probe=a690aee249
Hernan Imanol Rodriguez: https://linux-hardware.org/?probe=50993c10e6

5. ¿Qué diferencia existe entre un módulo y un programa?

Un módulo de kernel es un componente que se integra dinámicamente al núcleo del sistema operativo, ampliando sus funcionalidades (por ejemplo, drivers o extensiones del sistema). Opera en modo núcleo, con acceso directo a los recursos del hardware y a estructuras internas del sistema operativo. Su diseño está a cargo de un diseñador de sistemas, quien prioriza la eficiencia, estabilidad y seguridad del entorno operativo.

En contraste, un programa de usuario se ejecuta en modo usuario, sin privilegios directos sobre el hardware. Utiliza los servicios del sistema operativo mediante llamadas al sistema. Este tipo de software lo desarrolla un diseñador de programas o aplicaciones, enfocado en resolver problemas específicos del usuario final, con énfasis en la funcionalidad, usabilidad y mantenimiento.

En resumen:
El módulo de kernel extiende las capacidades internas del sistema operativo desde un nivel

privilegiado, mientras que el programa de usuario se apoya en esas capacidades para ejecutar tareas en un entorno seguro y controlado.

6. ¿Cómo puede ver una lista de las llamadas al sistema que realiza un simple helloworld en C?

Para observar las llamadas al sistema que realiza un programa en C (como un simple printf("Hello, world\n");), se puede utilizar la herramienta strace, que traza todas las llamadas al sistema realizadas por un proceso.

```
strace ./helloworld
```

Esta herramienta muestra en tiempo real todas las llamadas al sistema realizadas por el ejecutable helloworld, incluyendo aquellas asociadas a la escritura de pantalla, como write(), apertura de bibliotecas compartidas (openat()), asignación de memoria (mmap()), y finalización del programa (exit_group()), entre otras.

7 ¿Qué es un segmentation fault? ¿Cómo lo maneja el kernel y como lo hace un programa?

Un segmentation fault es un error que ocurre cuando un programa intenta acceder a una zona de memoria que no está permitida para tal programa, como acceder a una dirección nula o inválida o intentar escribir en una zona de memoria de solo lectura.
El kernel de Linux gestiona la memoria mediante la asignación de regiones protegidas para cada proceso. Cuando un programa accede de forma indebida a una dirección de memoria, el hardware genera una excepción de protección de memoria, que el kernel interpreta como una violación de segmentación. El kernel realiza:

1. El kernel detecta la violación mediante su mecanismo de manejo de fallos de página (page fault handler).
2. Marca el acceso como inválido.
3. Envía al proceso la señal SIGSEGV (Signal Segmentation Violation).
4. Si el programa no maneja esta señal, el kernel termina el proceso y puede generar un core dump para diagnóstico.

El segmentation fault es un mecanismo de protección de memoria que permite al sistema operativo mantener la integridad y seguridad de los procesos. El kernel lo detecta y actúa para prevenir daños, mientras que el programa rara vez tiene un manejo específico más allá de su corrección en desarrollo.

8. ¿Se animan a intentar firmar un módulo de kernel ? y documentar el proceso ?
Para firmar un módulo del kernel, primero debemos compilar el módulo del kernel. Luego creamos una Machine Owner Key (MOC).

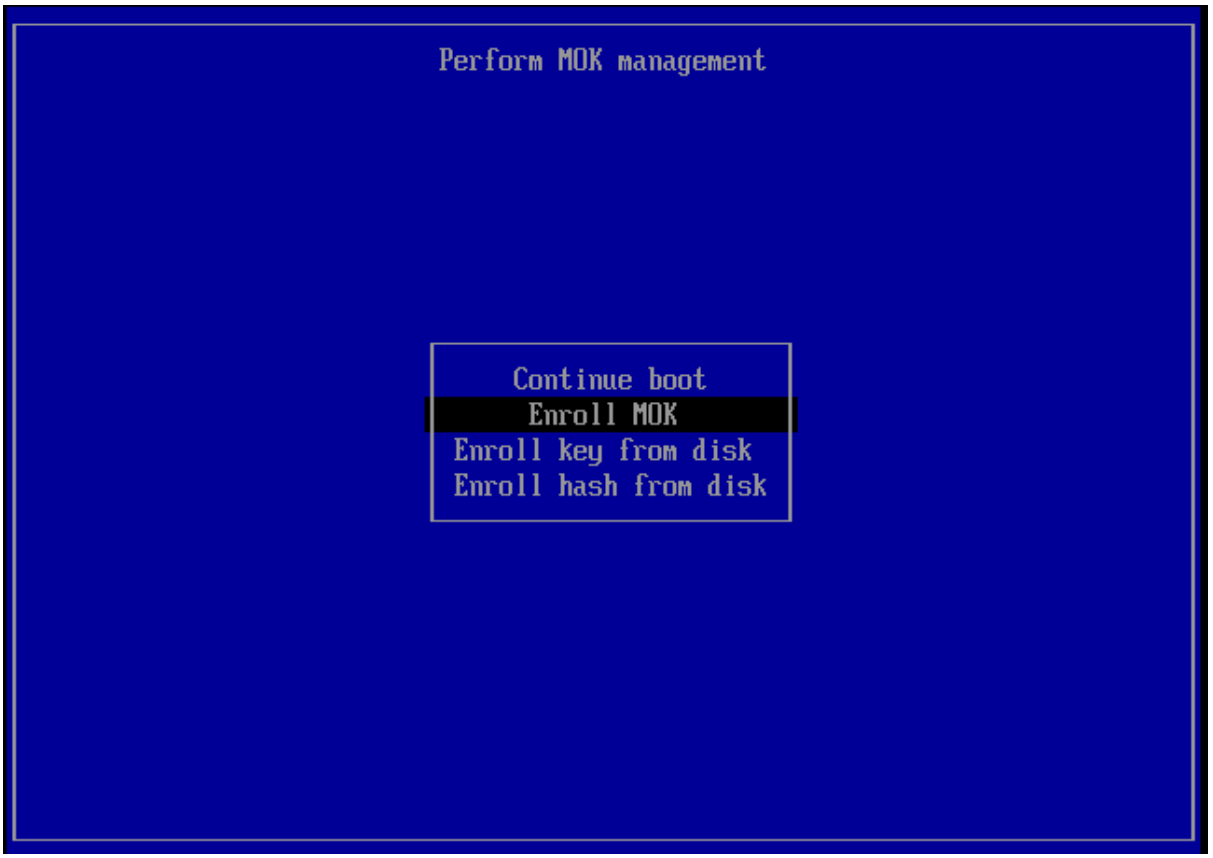
```
openssl req -new -x509 -newkey rsa:2048 -keyout MOK.priv -outform DER
```

```
-out MOK.der -days 36500 -subj "/CN=My Name/" -nodes
```

En este paso se nos solicita ingresar una contraseña para agregar la MOK en la UEFI.

```
mokutil --import MOK.der
```

Luego, una vez cargada la MOK se debe reiniciar la PC para registrar la clave en la UEFI mediante la ventana de administración de MOK.



Seguimos las instrucciones en pantalla y colocamos la password que definimos anteriormente.

Finalmente, firmamos el driver con

```
/usr/src/linux-headers-$(uname -r)/scripts/sign-file sha256
/root/MOK.priv /root/MOK.der modulo.ko
```

donde MOK.priv es la llave privada y MOK.der es la llave pública.

```
facundog@FacundoNotebook:~/kenel-modules/part1/module$ modinfo mimodulo.ko
filename:       /home/facundog/kenel-modules/part1/module/mimodulo.ko
author:         Catedra de SdeC
description:    Primer modulo ejemplo
license:        GPL
srcversion:     C6390D617B2101FB1B600A9
depends:
retpoline:      Y
name:           mimodulo
vermagic:       6.11.0-26-generic SMP preempt mod_unload modversions
sig_id:         PKCS#7
signer:         My Name
sig_key:        11:1C:E6:72:E7:FE:A2:5C:2A:A1:1E:BE:C7:E9:2D:E0:D1:14:6A:1F
sig_hashalgo:   sha256
signature:      69:D1:3A:F7:64:27:CA:D7:BF:AA:0F:90:B2:E7:E2:13:73:29:8C:5D:
                02:9F:22:45:31:8C:B7:82:D2:AE:4A:D8:EF:57:91:E0:C2:BA:8A:E7:
                4D:71:2C:95:B2:F5:ED:7A:1A:64:60:F2:1A:58:72:46:CB:61:EC:81:
                20:F9:A3:8C:E3:E7:5B:22:39:2A:0A:53:BE:93:96:C0:29:52:C6:72:
                B9:BA:40:23:F9:8A:89:88:CD:1D:A3:14:AC:86:58:58:19:3F:94:68:
                7D:9B:8D:9B:2A:11:AF:59:83:DB:98:64:32:9E:55:7E:66:9E:63:F3:
                B4:4F:70:C3:7C:65:DC:2F:E3:8B:0C:2E:9E:37:92:EA:E8:67:37:D4:
                E1:93:A6:0B:B9:B1:C2:AC:81:99:BB:0A:95:EE:3A:76:2E:81:42:E4:
                A2:A0:25:45:A6:52:B4:0B:55:3D:69:49:67:9B:41:F0:ED:00:48:35:
                79:A8:DF:C0:51:28:C5:1C:94:A7:9A:76:96:D3:F0:2B:27:C0:D3:E3:
                1C:34:7A:17:A0:42:9C:56:BA:39:61:F3:00:B3:51:69:86:C2:26:3A:
                DE:8F:B6:2E:68:BB:9F:5A:65:21:23:39:35:9A:C3:88:55:4B:11:0D:
                F6:03:6E:78:33:44:8F:D3:25:50:72:52:2F:BE:40:BB
```

9. Agregar evidencia de la compilación, carga y descarga de su propio módulo imprimiendo el nombre del equipo en los registros del kernel.

Para realizar la compilación de un módulo, se debe especificar al compilador que el codigo será a nivel kernel , también debemos importar las librerías necesarias. A continuación un ejemplo simple de una configuración Makefile:

```
obj-m += hello-module.o
PWD := $(CURDIR)

all:

	$(MAKE) -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
	$(MAKE) -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

y la respectiva carga y descarga, demostrando que el módulo se ha cargado efectivamente en los registros del kernel.

```
    imanol@fedora | ~/modulos                                                    ✔ | 21:36:34
❯ modinfo hello-module.ko
filename:       /home/imanol/modulos/hello-module.ko
license:        GPL
depends:
name:           hello_module
retpoline:      Y
vermagic:       6.14.6-200.fc41.x86_64 SMP preempt mod_unload
    imanol@fedora | ~/modulos                                                    ✔ | 22:13:54
❯ sudo insmod hello-module.ko
[sudo] contraseña para imanol:
    imanol@fedora | ~/modulos                                                    ✔ | 5s | 22:14:09
❯ sudo rmmod hello-module.ko
    imanol@fedora | ~/modulos                                                    ✔ | 22:14:24
❯ journalctl --since "1 hour ago" | grep kernel
may 25 21:15:34 fedora kernel: usb 2-4: current rate 13912564 is different from the runtime rate 16000
may 25 21:23:16 fedora kernel: usb 2-4: 3:1: cannot get min/max values for control 2 (id 3)
may 25 22:14:09 fedora kernel: Hello world 1.
may 25 22:14:24 fedora kernel: Goodbye world 1.
    imanol@fedora | ~/modulos
```

10. ¿Que pasa si mi compañero con secure boot habilitado intenta cargar un módulo firmado por mi?

Linux impide cargar el módulo firmado por un compañero.

```
facundog@FacundoNotebook:~/Documents/sdec-kernel-module$ cd firma-modulo/tomascorbalan/
facundog@FacundoNotebook:~/Documents/sdec-kernel-module/firma-modulo/tomascorbalan$ insmod modulo_firmado.ko
insmod: ERROR: could not insert module modulo_firmado.ko: Operation not permitted
facundog@FacundoNotebook:~/Documents/sdec-kernel-module/firma-modulo/tomascorbalan$
```

Esto es porque al estar activado el secure boot, el kernel impide cargar módulos firmados con llaves que no estén registradas.

```
o
[sudo] password for facundog:
insmod: ERROR: could not insert module modulo_firmado.ko: Invalid module format
facundog@FacundoNotebook:~/Documents/sdec-kernel-module/firma-modulo/tomascorbalan$
```

11. Dada la siguiente nota
https://arstechnica.com/security/2024/08/a-patch-microsoft-spent-2-years-preparing-is-making-a-mess-for-some-linux-users/

¿Cuál fue la consecuencia principal del parche de Microsoft sobre GRUB en sistemas con arranque dual (Linux y Windows)?

El parche de Microsoft, diseñado para abordar una vulnerabilidad en GRUB que permitía eludir las protecciones de Secure Boot, introdujo cambios en las políticas de Secure Boot Advanced Targeting (SBAT). Estos cambios revocaron certificados utilizados por versiones anteriores de GRUB, lo que provocó que, en sistemas con arranque dual, al intentar iniciar Linux, se mostrara un mensaje de error indicando una "violación de la política de seguridad" . Esto impidió que los usuarios pudieran arrancar sus sistemas Linux, afectando distribuciones como Ubuntu, Debian y Linux Mint.

¿Qué implicancia tiene desactivar Secure Boot como solución al problema descrito en el artículo?

Desactivar Secure Boot en la configuración del BIOS/UEFI permite que el sistema arranque sin verificar las firmas digitales de los cargadores de arranque, lo que puede ser una solución temporal para restaurar el acceso a Linux. Sin embargo, esta acción reduce la seguridad del sistema, ya que Secure Boot está diseñado para prevenir la ejecución de software no autorizado durante el proceso de arranque . Además, desactivar Secure Boot puede afectar características de seguridad en Windows, como BitLocker, que dependen de esta función para proteger la integridad del sistema

¿Cuál es el propósito principal del Secure Boot en el proceso de arranque de un sistema?

Secure Boot es una característica de seguridad que verifica que el software de arranque, como los cargadores de arranque y los controladores del sistema operativo, tenga firmas digitales válidas y esté autorizado por el fabricante del equipo. Su objetivo principal es prevenir la ejecución de software malicioso o no autorizado durante el proceso de arranque, protegiendo así la integridad del sistema desde las etapas iniciales