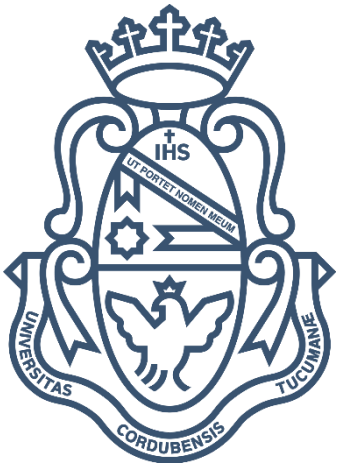

Universidad Nacional de Córdoba

Facultad de Ciencias Exactas, Físicas y Naturales

Sistemas de computación

TP4

Grupo:

Epsilon

Profesores:

Jorge, Javier Alejandro

Lamberti, Germán Andrés

Solinas, Miguel Ángel

Alumnos:

Campos, Mariano

Erlicher, Ezequiel

González, Damián Marcelo

NOTA: Link a uno de los repositorios (luegos los otros contienen lo mismo, y fueron forked), conteniendo este informe, y también archivos solicitados, como por ej. el del punto 2 y 9 del Cuestionario.

Desafío #1

- ¿Qué es *checkinstall* y para qué sirve?

Es una aplicación que reemplaza a `*make install*`. Mientras `*make install*` instala las cosas y despliega los archivos donde los *Makefile* lo indiquen, *checkinstall* por defecto instala lo intencionado mediante el **sistema de gestión de paquetes** propio de la distribución GNU/Linux actual. Por ejemplo, si estás trabajando en Debian, creará e instalará un *.deb.

Es posible pasarle la opción *–install=no*, en cuyo caso se saltará la parte de instalar mediante el sistema de gestión de paquetes del sistema, pero sí generará el respectivo binario (*.deb, siguiendo con el ejemplo). Luego este *.deb se puede compartir a otros usuarios de Debian, y estos pueden fácilmente instalar el software via su sustema de gestión de paquetes del sistema. Esta última forma de proceder se conoce como **“empaquetar un programa**”.

Fuente principal: https://wiki.debian.org/CheckInstall.

- Utilización, empaquetando un programa *hello world*.

Procedimos a crear un *Makefile*, que compile nuestro programa *hello_humans.c*. Posteriormente, luego de utilizar `*make*`, corrimos `*sudo checkinstall*`, ello nos pidió ingresar la descripción del paquete a generar e instalar, y después de pedirnos confirmación, mostrándonos los metadatos que llevaría el paquete a construir, procedió a crear e instalar el paquete *.deb.

Todos los archivos mencionados, se encuentran dentro de la carpeta **desafio-1** del repositorio vinculado a este informe. A continuación mostramos las imágenes del proceso.

```
ezerlich@ezerlich-Lenovo-V330-15IKB:~/Documents/TPS-SDC-2025/SDC-TP4-Epsilon/desafio-1$ make
gcc -Wall -c hello_humans.c
gcc -Wall -o hello_humans hello_humans.o
ezerlich@ezerlich-Lenovo-V330-15IKB:~/Documents/TPS-SDC-2025/SDC-TP4-Epsilon/desafio-1$ sudo checkinstall
[sudo] password for ezerlich:

checkinstall 1.6.3, Copyright 2010 Felipe Eduardo Sanchez Diaz Duran
           This software is released under the GNU GPL.


The package documentation directory ./doc-pak does not exist.
Should I create a default set of package docs?  [y]: y

Preparing package documentation...OK

*** No known documentation files were found. The new package
*** won't include a documentation directory.

Please write a description for the package.
End your description with an empty line or EOF.
>> "This is a simple hello humans program"
>> EOF
>>
```

Ejecución de make y checkinstall.

```
**************************************
**** Debian package creation selected ***
**************************************

This package will be built according to these values:

0 -  Maintainer: [ root@ezerlich-Lenovo-V330-15IKB ]
1 -  Summary: [ "This is a simple hello humans program" ]
2 -  Name:    [ desafio ]
3 -  Version: [ 1 ]
4 -  Release: [ 1 ]
5 -  License: [ GPL ]
6 -  Group:   [ checkinstall ]
7 -  Architecture: [ amd64 ]
8 -  Source location: [ desafio-1 ]
9 -  Alternate source location: [  ]
10 - Requires: [  ]
11 - Recommends: [  ]
12 - Suggests: [  ]
13 - Provides: [ desafio ]
14 - Conflicts: [  ]
15 - Replaces: [  ]
16 - Prerequires: [  ]
```

Ejecución de checkinstall, parte 2.

```
Enter a number to change any of them or press ENTER to continue:

Installing with make install...

========================= Installation results ==========================
install -Dm755 hello_humans /usr/local/bin/hello_humans

======================== Installation successful =========================

Copying files to the temporary directory...OK

Stripping ELF binaries and libraries...OK

Compressing man pages...OK

Building file list...OK

Building Debian package...OK

Installing Debian package...OK

Erasing temporary files...OK

Deleting temp dir...OK
```

Ejecución de checkinstall, parte 3.

```
Building file list...OK

Building Debian package...OK

Installing Debian package...OK

Erasing temporary files...OK

Deleting temp dir...OK

**********************************************************************

 Done. The new package has been installed and saved to

 /home/ezerlich/Documents/TPS-SDC-2025/SDC-TP4-Epsilon/desafio-1/desafio_1-1_amd64.deb

 You can remove it from your system anytime using:

      dpkg -r desafio

**********************************************************************

ezerlich@ezerlich-Lenovo-V330-15IKB:~/Documents/TPS-SDC-2025/SDC-TP4-Epsilon/desafio-1$
```

Ejecución de checkinstall, parte 4.

- **¿Qué acciones podrían impulsarse, que permitan mejorar la seguridad del Kernel?**

Es posible que potenciales módulos a cargarse en el kernel sean maliciosos. Un ejemplo de ello son los **rootkit**, un conjunto de herramientas maliciosas, diseñadas para ocultar la presencia de **malware** o accesos no autorizados en un sistema, incluso al administrador.

Un proceder mínimo que debiera hacerse siempre al cargar un módulo es asegurarse de que el módulo a cargar se encuentre **firmado**. Para módulos propios custom, por ejemplo, en sistemas que corren RHEL 8, este es el proceder para el autofirmado.

Desafío #2

- ¿Qué funciones tiene disponible un *programa*, y un módulo?

Un módulo de kernel tiene acceso a funciones y recursos **privilegiados del kernel**, que un programa de *espacio de usuario* no puede usar directamente. Por ejemplo, funciones de:

- Acceso directo al hardware (sin pasar por capas de abstracción).
- Gestión de memoria física y páginas (*kmalloc()*, *vmalloc(),* etc.).
- Acceso al espacio de direcciones del kernel.
- Control de interrupciones y manejo de dispositivos (*request_irq()*, etc.).
- Interaccción con subsistemas del kernel, como redes, sistema de archivos, proceso, etc.
- Definir llamadas al sistema nuevas o modificar las existentes.

Del otro lado de la vereda, un *programa de usuario* sólo puede usar las API expuestas por el kernel (syscalls, i.e.: funciones de **glibc**).

- Diferencias entre *espacio de usuario* y *espacio de Kernel*.

Característica	Espacio de Usuario	Espacio de Kernel
Acceso al hardware	No directo (usa syscalls)	Directo
Privilegios	Restringidos	Totales (modo supervisor)
Estabilidad	Un fallo afecta sólo al proceso	Un fallo puede colgar todo el sistema
Memoria accesible	Solo su propio espacio de memoria	Toda la memoria del sistema
Ejemplos típicos	Bash, Firefox, GCC, etc.	Drivers, scheduler, VFS, etc.
Modo de CPU	Modo usuario	Modo kernel
Comunicación	Vía llamadas al sistema (*syscalls*)	Interna y directa

- ¿Qué es un *espacio de datos*?

Refiere a un espacio de memoria, en donde se guardan **símbolos y datos de variables globales, y estáticas**, ya sea que tengan datos inicializados o no inicializados. No mencionamos variables *locales*, ya que ellas viven en el stack, cuando les *toca su momento*.

- ¿Qué son los *drivers*? Investigar contenido de */dev*.

Los drivers (*aka controladores de dispositivo*) son *programas* **del kernel**, que permiten que el SO se comunique con el HW. Cada dispositivo de HW (teclados, discos, placas de red, etc.) tienen su respectivo driver, para un respectivo SO. Son *intérpretes* entre el hardware y el software.

El directorio */dev* expone interfaces hacia diferente hardware contenido en el computador, de manera que programas puedan utilizarlos fácilmente. Por ejemplo, un disco rígido (*que se vincula con el software mediante su driver de almacenamiento*) está expuesto a ser accedido a través de */dev/sda*, un determinado puerto USB estará expuesto por ejemplo en */dev/ttyUSB0*, etc. Vale aclarar que existe el especial */dev/null*, que se puede utilizar, por ejemplo, para redirigir la salida estándar de un programa hacia allí, descartando así toda dicha salida, aunque sin generar ningún tipo de problema/error.

Procedimiento práctico

```
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules$ cd part1
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1$ cd module/
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ make
make -C /lib/modules/5.15.0-140-generic/build M=/home/dmg/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module modules
make[1]: Entering directory '/usr/src/linux-headers-5.15.0-140-generic'
  CC [M]  /home/dmg/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module/mimodulo.o
  MODPOST /home/dmg/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module/Module.symvers
  CC [M]  /home/dmg/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module/mimodulo.mod.o
  LD [M]  /home/dmg/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module/mimodulo.ko
  BTF [M] /home/dmg/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module/mimodulo.ko
Skipping BTF generation for /home/dmg/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module/mimodulo.ko due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-5.15.0-140-generic'
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ ls
Makefile  mimodulo.c  mimodulo.ko  mimodulo.mod  mimodulo.mod.c  mimodulo.mod.o  mimodulo.o  modules.order  Module.symvers
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ sudo insmod mimodulo.ko
[sudo] password for dmg:
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ sudo dmesg
[    0.000000] microcode: microcode updated early to revision 0x21, date = 2019-02-13
[    0.000000] Linux version 5.15.0-140-generic (buildd@lcy02-amd64-024) (gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0, GNU ld (GNU Binutil
s for Ubuntu) 2.38) #150-Ubuntu SMP Sat Apr 12 06:00:09 UTC 2025 (Ubuntu 5.15.0-140.150-generic 5.15.179)
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.15.0-140-generic root=UUID=3f6ee383-7f64-422d-a543-e0806ae0bace ro quiet splash
[    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:
[    0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000000009ebff] usable
[    0.000000] BIOS-e820: [mem 0x000000000009ec00-0x000000000009ffff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000000e0000-0x00000000000fffff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000000100000-0x00000000bfffffff] usable
```

Procedimiento práctico, parte 1: compilación e instalación de módulo custom.

```
[13797.424027] mimodulo: module verification failed: signature and/or required key missing - tainting kernel
[13797.424399] Modulo cargado en el kernel.
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ lsmod | grep mod
mimodulo               16384  0
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ sudo rmmod mimodulo
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ sudo dmsg
sudo: dmsg: command not found
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ sudo dmesg
[    0.000000] microcode: microcode updated early to revision 0x21, date = 2019-02-13
[    0.000000] Linux version 5.15.0-140-generic (buildd@lcy02-amd64-024) (gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0, GNU ld (GNU Binutil
s for Ubuntu) 2.38) #150-Ubuntu SMP Sat Apr 12 06:00:09 UTC 2025 (Ubuntu 5.15.0-140.150-generic 5.15.179)
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.15.0-140-generic root=UUID=3f6ee383-7f64-422d-a543-e0806ae0bace ro quiet splash
[    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:
[    0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000000009ebff] usable
[    0.000000] BIOS-e820: [mem 0x000000000009ec00-0x000000000009ffff] reserved
```

Procedimiento práctico, parte 2: remoción de módulo custom.

```
[13797.424027] mimodulo: module verification failed: signature and/or required key missing - tainting kernel
[13797.424399] Modulo cargado en el kernel.
[13858.888905] Modulo descargado del kernel.
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ lsmod | grep mod
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ cat /proc/modules | grep mod
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ modinfo mimodulo.ko
filename:       /home/dmg/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module/mimodulo.ko
author:         Catedra de SdeC
description:    Primer modulo ejemplo
license:        GPL
srcversion:     C6390D617B2101FB1B600A9
depends:
retpoline:      Y
name:           mimodulo
vermagic:       5.15.0-140-generic SMP mod_unload modversions
```

Procedimiento práctico, parte 3: `modinfo` de nuestro módulo custom.

```
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ modinfo /lib/modules/$(uname -r)/kernel/crypto/des_generic.ko
filename:       /lib/modules/5.15.0-140-generic/kernel/crypto/des_generic.ko
alias:          crypto-des3_ede-generic
alias:          des3_ede-generic
alias:          crypto-des3_ede
alias:          des3_ede
alias:          crypto-des-generic
alias:          des-generic
alias:          crypto-des
alias:          des
author:         Dag Arne Osvik <da@osvik.no>
description:    DES & Triple DES EDE Cipher Algorithms
license:        GPL
srcversion:     E943DEED5E78CA63B2A81B5
depends:        libdes
retpoline:      Y
intree:         Y
name:           des_generic
vermagic:       5.15.0-140-generic SMP mod_unload modversions
sig_id:         PKCS#7
signer:         Build time autogenerated kernel key
sig_key:        0F:EF:28:01:ED:33:9F:9A:C1:E0:5E:82:E3:F8:1E:C0:16:5C:23:A8
sig_hashalgo:   sha512
signature:      77:57:D1:B3:54:42:4D:B0:26:F4:E6:E7:D6:88:E8:8F:88:DE:F6:9B:
```

Procedimiento práctico, parte 4: `modinfo` de módulo integrado en el kernel.

Cuestionario

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

Las diferencias son (observables entre imágenes *parte 3* y *parte 4* precedentes):

- **name**: nombre de módulo.
- **srcversion**: es una huella digital (hash) que representa la versión exacta del código fuente del módulo al momento de compilarlo. Se calcula a partir del contenido del archivo fuente del módulo (y a veces sus dependencias) mediante un hash SHA1.
- **description**: descripción del módulo.
- **author**: autor del módulo.
- **filename**: nombre de archivo del módulo.
- No existe firma en el módulo custom.

2. ¿Qué drivers/modulos 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.

Por listar algunos ejemplos de drivers/módulos cargados en nuestras PCs (laptops), tenemos:

- **i915**: Controlador de GPU integrado para gráficos Intel en el kernel de Linux. Soporta las familias de chips gráficos Intel HD, UHD, Iris y similares. Este módulo está siendo utilizado por 23 otros módulos en la PC de Damián, por 15 en la PC de Ezequiel, y en el caso de Mariano, utiliza **amdgpu**, que es análogo a i915, utilizado por 35 otros módulos.
- **drm**: Direct Rendering Manager. Es un subsistema del kernel de Linux que se encarga de la gestión de gráficos modernos, principalmente gráficos acelerados por hardware. Es una interfaz entre el hardware gráfico (la GPU) y el sistema de ventanas (como X11). El módulo **drm** no dibuja gráficos directamente, sino que proporciona **servicios de bajo nivel** para otros controladores. En la PC de Damián está siendo utilizado por 11 otros módulos (entre ellos **i915**); en la PC de Ezequiel, en cambio, este módulo está *particionado* en otros 2 submódulos (**drm_buddy**, y **drm_display_helper**); por último en la PC de Mariano, sucede algo similar a la de Ezequiel, el módulo **drm** se encuentra *particionado* en otros 4 submódulos (**drm_buddy**, **drm_exec**, **drm_suballoc_helper**, y **drm_ttm_helper**).
- **btusb**: Permite usar adaptadores Bluetooth conectados por USB. A pesar de que la mayoría de Notebooks, tiene la tecnología Bluetooth embebida dentro de la carcasa de una laptop, internamente, ese hardware está conectado via bus USB al resto de la PC, y es por esa razón, que este módulo se encuentra cargado en PC de Damián, no siendo utilizado en el momento de la captura, pero sí cargado y listo para ser usado. En la PC de Ezequiel, y Mariano sucede lo mismo.

A continuación presentamos las diferencias (*diffs*) entre cada output de `*lsmod > modulos_cargados.txt*`:

- Diffs *lsmod* Damián vs Ezequiel.
- Diffs *lsmod* Damián vs Mariano.
- Diffs *lsmod* Ezequiel vs Mariano.

3. ¿Cuales no están cargados pero están disponibles? ¿Que pasa cuando el driver de un dispositivo no está disponible?

Como se puede observar en la siguiente imagen, hay 6030 módulos disponibles, de los cuales hay cargados 136 en esta PC en particular.

```
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ find /lib/modules/$(uname -r) -type f -name '*.ko*' | wc -l
6030
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$
```

Output, cantidad de módulo disponibles a ser cargados (algunos de ellos, ya cargados).

Cuando el driver de un dispositivo no está disponible sucede lo siguiente:

- El dispositivo no funciona correctamente porque el sistema operativo no puede comunicarse con él.
- No se crean entradas en */dev* ni aparece en herramientas como *ifconfig*, *lsblk*, o *lspci -k* (el cual puede indicar que no hay módulo asociado).
- El comando dmesg puede mostrar mensajes indicando que no se encontró un controlador para el dispositivo.
- *lspci -k* puede listar el hardware, pero con la leyenda "*Kernel modules: <none>*".
- Al intentar cargar el módulo con modprobe o insmod, el sistema puede devolver errores del tipo: "*Module not found*".
- Esto ocurre cuando el módulo no está presente en el sistema, ni como módulo separado ni integrado en el kernel.

Posibles soluciones:

- Instalar el paquete del sistema que contiene módulos adicionales.
- Compilar manualmente el módulo desde el código fuente del kernel o del fabricante.
- Cambiar a una versión del kernel que incluya soporte para el dispositivo.

4. Correr *hwinfo* en una PC real con HW real y agregar la URL de la información de HW en el reporte.

```
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ sudo -E hw-probe -all -upload
WARNING: 'edid-decode' package is not installed
Probe for hardware ... Ok
Reading logs ... Ok
Uploaded to DB, Thank you!

Probe URL: https://linux-hardware.org/?probe=ca1a648a8f
```

hwinfo (hw-probe) corrido en una de nuestras PC.

Aquí está el vínculo al reporte.

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

La diferencia principal es el lugar donde se ejecutan (*kernel space* vs *user space*, respectivamente). Aquí mas detalles:

Módulo (del kernel):

- Se ejecuta dentro del kernel, es decir, en modo núcleo (kernel space).
- Tiene acceso directo al hardware y a recursos protegidos del sistema.
- Es parte del sistema operativo y puede afectar la estabilidad y seguridad si falla.
- Se carga y descarga dinámicamente con herramientas como modprobe o insmod.
- Ejemplo: un módulo controlador de un disco, tarjeta de red, o sistema de archivos.

Programa de usuario:

- Se ejecuta en modo usuario (user space), aislado del kernel.
- No tiene acceso directo al hardware ni a recursos críticos, debe pedir al kernel vía llamadas al sistema (syscalls).
- Si falla, afecta sólo al programa, no al sistema completo.
- Son las aplicaciones normales: navegadores, editores de texto, utilidades, scripts, etc.
- Se ejecutan como procesos independientes con menor privilegio.

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

La forma mas común y práctica es utilizando la herramienta **strace** de Linux. Suponiendo que nuestro programa compilado y ejecutable es *helloworld.exe*, luego se hace `*strace ./helloworld.exe*` lo cual imprimirá en pantalla todas las llamadas al sistema (syscalls) que el programa hace en su ejecución. Lo que hace es interceptar y registrar las llamadas al sistema que realiza un proceso. Es útil para depurar , entender qué hace un programa a bajo nivelo qué archivos, dispositivos, etc. utiliza.

Por ejemplo, en un programa tan simple como este:

```c
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

Simple programa "helloworld".

El output de `*strace*` podría ser:

```
execve("./hello", ["./hello"], 0x7ffd2a9f6828 /* 30 vars */) = 0
brk(NULL)                               = 0x55a3c4a15000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8d1c2b1000
write(1, "Hello, World!\n", 14)         = 14     # escribe en stdout (fd=1)
exit_group(0)                           = ?
```

Respectivo `strace` output para el programa anterior.

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

Un *segmentation fault* (fallo de segmentación) es un error que ocurre cuando un programa intenta acceder a una zona de memoria que no tiene permitido leer o escribir, por ej.:

- Leer o escribir en una dirección de memoria inválida o no asignada.
- Acceder a memoria protegida (como áreas del sistema o memoria de otro proceso).
- Desreferenciar un puntero nulo o corrupto.

¿Cómo lo maneja el kernel?

- Cuando el CPU detecta un acceso inválido a memoria, genera una excepción de protección (*fault*).
- El kernel recibe esta excepción y la interpreta como un *SIGSEGV* (*signal segmentation violation*).
- El kernel envía la señal *SIGSEGV* al proceso que causó la violación.
- Si el proceso no tiene un manejador específico para *SIGSEGV*, el kernel termina el proceso (core dump opcional).
- El sistema operativo puede generar un archivo core dump que contiene el estado de la memoria y CPU para ayudar a diagnosticar el fallo.

¿Cómo lo maneja un programa de usuario?

- Por defecto, el proceso termina abruptamente cuando recibe *SIGSEGV*.
- El programa puede instalar un handler para *SIGSEGV* usando funciones como *signal()* o *sigaction()* para ejecutar código personalizado (por ejemplo, limpiar recursos, registrar logs).
- Sin embargo, generalmente no se puede continuar la ejecución normal porque el fallo indica un problema grave de acceso a memoria.

8. ¿Se animan a intentar firmar un módulo de kernel, y documentar el proceso?

```
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ openssl req -new -x509 -newkey rsa:2048 -keyout MOK.priv -outform DER -out MOK.der -nodes -days 3650
0 -subj "/CN=ModTest/"
.....+..+...+.+.....+.....++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*..+...+....+...+..+.+.+.....+...+.............+..+.............+
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*.+.....+....+.....+.+...+..+.+...+...........+.....+....+...+.....+.....+.....+..+.....
.+...+.............+...+...+.+.............+.+...+..........+.+.....+.+.....+.....................+....+.....+....+...+.............+.....+.....+...
.+.....+.....+...+.....+...+....+...+....+..+....+..+.....+...+..+.....++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.....+..+....+.....+...+..+...+.............+.+.....++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*.....+++++++++++++++++++++++++++++++
++++++++++++++++++++++++++*..+.....+..+.....+.............+....+..+...+.....................+.....+..+.............+.....+.....+..+....+.....+.....+
.....+.....+....+....+.+.....+.....+.....+.....................+.+.....+.+.....+...+.+..+...+.....+.....+....+....+....+....+..+.+.....+...+
..+...+...+...+.+.....+.....+.+..+...........+...+...+.....+.....+...+....+.....+.....+...+.....+.....+...+.....+...+.....+....+...++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-----
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ ls
Makefile  mimodulo.c  mimodulo.ko  mimodulo.mod  mimodulo.mod.c  mimodulo.mod.o  mimodulo.o  modules.order  Module.symvers  MOK.der  MOK.priv
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ /usr/src/linux-headers-$(uname -r)/scripts/sign-file sha256 MOK.priv MOK.der mimodulo.ko
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ modinfo mimodulo.ko
filename:       /home/dmg/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module/mimodulo.ko
author:         Catedra de SdeC
description:    Primer modulo ejemplo
license:        GPL
srcversion:     C6390D617B2101FB1B600A9
depends:
retpoline:      Y
name:           mimodulo
vermagic:       5.15.0-140-generic SMP mod_unload modversions
sig_id:         PKCS#7
signer:         ModTest
sig_key:        2B:5D:DC:45:58:30:0A:0A:06:DE:61:37:CF:9B:5E:CB:D9:03:4A:0C
sig_hashalgo:   sha256
signature:      9C:00:D4:23:89:63:DC:AB:C2:3C:3A:B6:77:2E:4A:9E:14:BE:F2:B7:
                BF:3A:CD:1B:35:29:D4:26:3E:EA:57:84:54:B6:01:A9:F2:3C:B2:37:
                AE:33:26:9B:21:EA:C9:48:19:91:FD:AC:F5:F1:B5:AB:F0:E1:F5:07:
                1C:EF:D7:74:0E:87:76:48:E2:8A:0B:CE:A7:A3:7B:6A:B5:4E:8A:47:
                04:44:98:BD:13:45:38:AD:08:58:E9:1F:1F:64:01:2E:9A:90:5B:30:
                F5:D0:8F:16:57:BD:BE:AF:C6:39:74:18:A2:C4:74:F1:6D:A8:77:FA:
                E6:A8:F5:16:1C:07:E2:75:17:42:40:93:EC:A9:3C:22:79:BD:67:CA:
                2A:6A:71:C9:A2:8C:06:87:63:87:EC:F8:19:29:B0:B2:FE:ED:34:AE:
                83:CE:B6:1E:43:F8:74:15:6D:6E:46:B8:FD:73:BC:1D:69:14:F0:C3:
                5B:01:00:C0:57:4F:A5:FD:76:E3:7D:AD:CF:6B:C6:09:40:ED:39:36:
                57:E7:4C:EC:36:DC:BD:B3:FF:38:DA:BB:4F:4C:84:04:FC:FE:ED:D9:
                0A:61:9E:F2:37:B7:EE:70:44:30:11:9E:BB:1C:2C:8C:D5:7C:51:CC:
                D1:C6:C1:47:4E:7E:0D:E3:DB:5A:02:EC:81:DF:F6:BA
```

Proceso de generación de firma, y efectivo firmado del módulo custom.

En la imagen anterior se pueden observar básicamente 3 pasos:

a. Generación de una clave *didáctica* (exentuada de cierta potencial rigurosidad).
b. Firmado del módulo.
c. Demostración de que el módulo fué efectivamente firmado.

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

Se modificó *mimodulo.c* pristino, de modo que ahora imprima el host que carga y descarga el módulo:

```c
#include <linux/utsname.h> /* Requerido para llamar a init_utsname() */
#include <linux/module.h> /* Requerido por todos los módulos */
#include <linux/kernel.h> /* Definición de KERN_INFO */
MODULE_LICENSE("GPL");  /*  Licencia del modulo */
MODULE_DESCRIPTION("Primer modulo ejemplo");
MODULE_AUTHOR("Catedra de SdeC");

/* Función que se invoca cuando se carga el módulo en el kernel */
int modulo_lin_init(void)
{
  printk(KERN_INFO "Modulo cargado en el kernel, desde %s.\n", init_utsname()->nodename);

  /* Devolver 0 para indicar una carga correcta del módulo */
  return 0;
}

/* Función que se invoca cuando se descarga el módulo del kernel */
void modulo_lin_clean(void)
{
  printk(KERN_INFO "Modulo descargado del kernel, desde %s.\n", init_utsname()->nodename);
}

/* Declaración de funciones init y exit */
module_init(modulo_lin_init);
module_exit(modulo_lin_clean);
```

Modificación en string impresa al cargar y descargar el módulo custom.

A continuación, las evidencias solicitadas:

```
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ ls
Makefile  mimodulo.c
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ make
make -C /lib/modules/5.15.0-140-generic/build M=/home/dmg/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module modules
make[1]: Entering directory '/usr/src/linux-headers-5.15.0-140-generic'
  CC [M]  /home/dmg/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module/mimodulo.o
  MODPOST /home/dmg/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module/Module.symvers
  CC [M]  /home/dmg/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module/mimodulo.mod.o
  LD [M]  /home/dmg/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module/mimodulo.ko
  BTF [M] /home/dmg/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module/mimodulo.ko
Skipping BTF generation for /home/dmg/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module/mimodulo.ko due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-5.15.0-140-generic'
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ sudo insmod mimodulo.ko
[sudo] password for dmg:
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ sudo dmesg
[    0.000000] microcode: microcode updated early to revision 0x21, date = 2019-02-13
[    0.000000] Linux version 5.15.0-140-generic (buildd@lcy02-amd64-024) (gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0, GNU ld (GNU Binuti
ls for Ubuntu) 2.38) #150-Ubuntu SMP Sat Apr 12 06:00:09 UTC 2025 (Ubuntu 5.15.0-140.150-generic 5.15.179)
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.15.0-140-generic root=UUID=3f6ee383-7f64-422d-a543-e0806ae0bace ro quiet splash
[    0.000000] KERNEL supported cpus:
[    0.000000]   Intel GenuineIntel
[    0.000000]   AMD AuthenticAMD
[    0.000000]   Hygon HygonGenuine
[    0.000000]   Centaur CentaurHauls
```

Compilación e instalación de módulo custom, versión nueva.

```
[19950.834624] wlp2s0: associated
[19950.897897] wlp2s0: Limiting TX power to 30 (30 - 0) dBm as advertised by 08:7e:64:8a:a4:68
[19950.917941] IPv6: ADDRCONF(NETDEV_CHANGE): wlp2s0: link becomes ready
[21453.592432] Modulo cargado en el kernel, desde dmg-BANGHO.
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ lsmod | grep mod
mimodulo               16384  0
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ sudo rmmod mimodulo
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ sudo dmesg
[    0.000000] microcode: microcode updated early to revision 0x21, date = 2019-02-13
[    0.000000] Linux version 5.15.0-140-generic (buildd@lcy02-amd64-024) (gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0, GNU ld (GNU Binuti
ls for Ubuntu) 2.38) #150-Ubuntu SMP Sat Apr 12 06:00:09 UTC 2025 (Ubuntu 5.15.0-140.150-generic 5.15.179)
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.15.0-140-generic root=UUID=3f6ee383-7f64-422d-a543-e0806ae0bace ro quiet splash
[    0.000000] KERNEL supported cpus:
[    0.000000]   Intel GenuineIntel
[    0.000000]   AMD AuthenticAMD
[    0.000000]   Hygon HygonGenuine
```

Remoción de módulo custom, previa evidencia de nueva str impresa.

```
[19950.834351] wlp2s0: RX AssocResp from 08:7e:64:8a:a4:68 (capab=0x1411 status=0 aid=5)
[19950.834624] wlp2s0: associated
[19950.897897] wlp2s0: Limiting TX power to 30 (30 - 0) dBm as advertised by 08:7e:64:8a:a4:68
[19950.917941] IPv6: ADDRCONF(NETDEV_CHANGE): wlp2s0: link becomes ready
[21453.592432] Modulo cargado en el kernel, desde dmg-BANGHO.
[21480.644585] Modulo descargado del kernel, desde dmg-BANGHO.
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$ lsmod | grep mod
dmg@dmg-BANGHO:~/Desktop/SDC-TP4-Epsilon/kenel-modules/part1/module$
```

Evidencia de impresión de nueva str, y evidencia de remoción efectiva.

10. ¿Qué pasa si mi compañero con Secure Boot habilitado intenta cargar un módulo firmado por mi?

El módulo será rechazado y no se cargará, a menos que:

- La clave con la que firmé está registrada en el sistema UEFI (o análogo) de su equipo como clave confiable (en el *keyring* de Secure Boot).
- Deshabilite Secure Boot.

11. Dada la siguiente nota:

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

El parche de seguridad de Microsoft, lanzado en agosto de 2024, introdujo una política de Secure Boot Advanced Targeting (SBAT) destinada a bloquear gestores de arranque vulnerables como GRUB2. Sin embargo, esta actualización se aplicó inadvertidamente a sistemas con arranque dual, provocando que muchos usuarios de Linux no pudieran iniciar sus sistemas y enfrentaran errores como "SBAT self-check failed: Security Policy Violation".

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

Desactivar Secure Boot permite que el sistema arranque sin verificar las firmas digitales de los componentes de arranque, lo que puede resolver el problema de incompatibilidad con GRUB. Sin embargo, esto también desactiva una capa importante de seguridad que protege contra *malware* que intenta ejecutarse durante el proceso de arranque. Por lo tanto, aunque es una solución temporal efectiva, compromete la seguridad del sistema y no se recomienda como solución permanente.

c. ¿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 la integridad y autenticidad de los componentes del proceso de arranque, asegurándose de que solo se ejecute software, y firmware firmados y confiables. Su objetivo principal es prevenir la ejecución de *malware* o software no autorizado durante el inicio del sistema, protegiendo así la integridad del SO desde las etapas más tempranas del arranque.